



WILEY EMERGING TECHNOLOGY SERIES

Full Stack Java Development with Spring MVC, Hibernate, jQuery, and Bootstrap

Mayur Ramgir

- ▶ **Starter Kit** of full stack application.
- ▶ **Codes** for all examples in the text.
- ▶ Instructive **Videos** to supplement the text.
- ▶ Possible **Interview Questions and Answers** inside.

WILEY

Full Stack Java Development with Spring MVC, Hibernate, jQuery, and Bootstrap

WILEY EMERGING TECHNOLOGY SERIES

Full Stack Java Development with Spring MVC, Hibernate, jQuery, and Bootstrap

Mayur Ramgir

Founder of EverythingTech and CEO of Zonopact, Inc

WILEY

Full Stack Java Development with Spring MVC, Hibernate, jQuery, and Bootstrap

Copyright © 2020 by Wiley India Pvt. Ltd., 4436/7, Ansari Road, Daryaganj, New Delhi-110002.

Cover Image: naddi/Getty Images

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or scanning without the written permission of the publisher.

Limits of Liability: While the publisher and the author have used their best efforts in preparing this book, Wiley and the author make no representation or warranties with respect to the accuracy or completeness of the contents of this book, and specifically disclaim any implied warranties of merchantability or fitness for any particular purpose. There are no warranties which extend beyond the descriptions contained in this paragraph. No warranty may be created or extended by sales representatives or written sales materials. The accuracy and completeness of the information provided herein and the opinions stated herein are not guaranteed or warranted to produce any particular results, and the advice and strategies contained herein may not be suitable for every individual. Neither Wiley India nor the author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Disclaimer: The contents of this book have been checked for accuracy. Since deviations cannot be precluded entirely, Wiley or its author cannot guarantee full agreement. As the book is intended for educational purpose, Wiley or its author shall not be responsible for any errors, omissions or damages arising out of the use of the information contained in the book. This publication is designed to provide accurate and authoritative information with regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services.

Trademarks: All brand names and product names used in this book are trademarks, registered trademarks, or trade names of their respective holders. Wiley is not associated with any product or vendor mentioned in this book.

Other Wiley Editorial Offices:

John Wiley & Sons, Inc. 111 River Street, Hoboken, NJ 07030, USA

Wiley-VCH Verlag GmbH, Pappelallee 3, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 1 Fusionopolis Walk #07-01 Solaris, South Tower, Singapore 138628

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada, M9W 1L1

First Edition: 2020

ISBN: 978-81-265-1991-0

ISBN: 978-81-265-8902-9 (ebk)

www.wileyindia.com

Printed at:

I dedicate this book to my wife for her love and support, my little 5-year-old son for all his love and affection, my mom for raising me despite difficulties, my late father who has been in heaven since I was 6 years old, my father-in-law and mother-in-law for all their love and support.

Preface

Today, applications are developed in order to work with mobile devices or on hosted web pages. This is a direct outcome of faster Internet speeds, wider Internet access, more users, and the introduction of powerful mobile devices. Therefore, we cannot develop applications by the same old way. Instead of just having an isolated desktop or mobile application, we need applications with a distributed back-end infrastructure which will serve a variety of front-end clients over the Internet. This is where full stack development comes into play. A full-stack engineer has the job to keep every single part of the system running smoothly. Traditionally, the development of a web application or website can be divided into three different categories – front-end, back-end, and database related processes. Previously, each of these departments was handled by individual developers. However, nowadays organizations demand a developer who can work equally well with both front-end and back-end technologies. Such developers with knowledge of all departments are known as full stack developers. This term can be applied for either software/app developers or web developers. Web development as a career choice has skyrocketed these days since every business and organization is trying to grow with the help of the Internet. Nowadays, very few businesses can grow without having a web presence.

Full stack developers are needed to provide organizations with the skillset and agility they need to succeed in the digital world. Therefore, a full stack developer must have specialization in taking a number of different tasks and work with skills from other disciplines. This does not mean mastering all the technologies. But the industry expects the professional to work on the client as well as server sides and understand the operations while developing an application.

To become a good full stack developer you will need to work on the basics first. This includes understanding the fundamentals of software engineering and the remaining will follow you accordingly. This is the perfect book for beginners, who want to learn about full stack development. This book will teach readers about full-stack development, from the basics to advance topics with HTML, CSS, JQuery, Bootstrap, Spring MVC, and Hibernate. Any individual without much programming knowledge can greatly benefit from this book and learn all about full stack development. However, it is also good for engineers in traditionally structured organizations who are aspiring to transform into this new digital world. The developers already on the full-stack teams can also use this book to refine their skills further.

Organization of the Book

Section 1 – Basic Building Blocks of Full Stack Development

Chapter 1 Introduction to Full Stack Development: This chapter introduces the fundamentals of full stack development to readers. The reader will be exploring the essential technologies and ecosystem of web application development. Model–View–Controller (MVC) and object-relational mapping (ORM) is introduced with JSON, XML, and Hibernate. You will learn about front-end development with web technologies such as HTML, JQuery, and Bootstrap. Lastly, web services are also explained with API-based architecture.

Chapter 2 Getting Started with Full Stack Development: A Project Idea: This chapter elucidates how to design a blueprint for the project with different entities. The readers will learn about design decisions with flowcharts and entity relationship diagram for the project. By the end of this chapter, readers will be able to design front-end and web services APIs.

Section 2 – Front-End Technologies

Chapter 3 Introduction to HyperText Markup Language: This chapter examines the fundamentals of programming with HyperText Markup Language (HTML). This chapter explores some real-life examples to understand the development of web pages and web applications.

Chapter 4 Introduction to Cascading Style Sheets: This chapter describes the fundamentals of Cascading Style Sheets (CSS). The readers will explore how CSS can describe HTML elements to be displayed on a screen. There are multiple interesting examples in this chapter to understand the functionality of CSS with HTML.

Chapter 5 Introduction to JQuery: This chapter is an in-depth study on jQuery and how it works as a JavaScript library. The readers will understand how it helps in simplifying JavaScript programming. There are multiple interesting examples of JQuery, which makes the learning experience much easier.

Chapter 6 Introduction to Bootstrap: This chapter discusses Bootstrap in detail. By reading this chapter, the readers will understand all about mobile-first front-end web development. There are numerous examples in this chapter which teaches about JavaScript-based templates, forms, buttons, navigation options, and other interface components.

Section 3 – Front-End Development

Chapter 7 Build Pages with HTML and CSS: This chapter details how to build responsive, interactive, and mobile-friendly web pages by using HTML and CSS. Building web pages will help the readers to learn how to build their own websites.

Chapter 8 Use of jQuery on HTML and CSS: This chapter describes the use of jQuery on the websites created by using HTML and CSS. It will be easier for readers to manipulate HTML DOM elements and add interactivity to the page after learning JQuery.

Chapter 9 Use of Bootstrap to Make HTML Responsive: This chapter will discuss how to use Bootstrap on the created webpages to make them mobile friendly. This greatly helps us to create more responsive websites.

Section 4 – Java Concepts

Chapter 10 Introduction to Java Language: This chapter focuses on the basics of Java and how it became one of the most used programming languages in the world. The reader will learn about the history and different functionalities of Java.

Chapter 11 Language Syntax and Elements of Language: This chapter discusses the sets of rules and regulations used while programming a Java program. These rules are very important to master the different elements of the Java programming language. It covers primitive data types, keywords, string options, wrapper classes, autoboxing, unboxing, various operators, etc.

Chapter 12 Object-Oriented Programming: This chapter introduces object-oriented programming (OOP) and its various associated concepts like object, class, inheritance, abstraction, encapsulation, polymorphism, etc. The reader will learn how to implement OOP in Java programs. It also covers abstract class, interface, overriding and overloading principals. At the end of this chapter, readers will be well aware of OOP concepts and can use effectively for writing Java programs.

Chapter 13 Generics and Collections: This chapter discusses generic programming and its applications in Java program. It introduces the concept like type system identification and how to use to create a safety net to avoid logical errors. It also teaches the concept of collection with various interfaces like map, set, list, queues, collection, ConcurrentNavigableMap, ConcurrentMap, SortedSet, SortedMap, NavigableMap, etc. It covers various examples to understand the concept in detail.

Chapter 14 Error Handling: This chapter examines error handling and logical errors. It also covers the concept of syntactical errors like capitalization, string split, missing or improper imports of classes, and missing curly braces. And concept of semantic errors such as improper use of operators, incompatible types, precision, and scoping. It teaches the use of try, catch and finally block with a lot of examples. Finally it covers the difference between checked and runtime exceptions.

Chapter 15 Garbage Collection: This chapter focuses on garbage collection, its purpose, and uses. It takes the readers through the concept of memory management. With a lot of examples, readers will be able to make objects eligible for collection.

Chapter 16 Strings, I/O Operations, and File Management: This chapter describes various string operations and the concept of StringBuffer and StringBuilder. Along with string, it teaches various I/O operations including InputStream and OutputStream. In the final part of the chapter, it covers file management like accessing and manipulating files.

Chapter 17 Data Structure and Integration in Program: This chapter covers various data structures and their uses in the program. Readers will get introduced to primitive and non-primitive data structures. It also highlights the difference between tree and graph. Readers will learn about binary tree and other tree data structures.

Chapter 18 Lambdas and Functional Programming: This chapter focuses on functional programming and how to use to incorporate in the Java program. Readers will learn about higher order and first class functions. It also covers pure functions and their characteristics. Readers will explore recursion functions, evaluation, and referential transparency. It covers lambdas and working examples of all functions using Java.

Chapter 19 Multithreading and Reactive Programming: This chapter covers multithreading and reactive programming. Readers will learn how to work with multiple threads and get introduced to the concept of deadlock, race conditions, synchronization, etc. Readers will also learn about concurrency API improvements, how to design concurrent program, control sequential execution, and avoid lazy initialization.

Section 5 – Spring and Hibernate Concepts

Chapter 20 Introduction to Spring and Spring Model–View–Controller: This chapter discusses spring framework. It explains what is spring and basic features of spring like inversion of control and dependency injection. Readers will also learn about bean scopes like singleton and prototype. It explains Spring MVC in detail where readers learn about role of controller and dispatcher servlet.

Chapter 21 Introduction to Hibernate: This chapter examines database operations through an object-relational mapping (ORM). Readers will get introduced to the concept of ORM and its benefits. Readers will learn about Hibernate and how to use it. Database configuration will be covered as well. Readers will get exposed to various annotations, inheritance mapping, etc. Readers will also learn how to map class to table and perform CRUD operations. The chapter also covers the concept of transactions and caching along with Hibernate Query language and named query.

Section 6 – Back-End Development

Chapter 22 Develop Web Services for the APIs: This chapter focuses on developing web services. Readers will learn how to create a Spring MVC project using Spring Boot along with REST webservice endpoints. Readers will see the practical use of MVC framework using Eclipse based IDE like Spring Tool Suite. Chapter also covers the Data Access Object (DAO) concept and object design pattern. Readers will learn about useful annotations like @Autowired, @RequestMapping, and @PathVariable.

Chapter 23 Develop Models with Hibernate: This chapter focuses on creating hibernate models and database access for the demo application. Readers will know how to perform CRUD operation on database using Data Access Object (DAO). Chapter also covers methods save and saveOrUpdate.

Annexures

Annexure A Consuming Web Services: This annexure discusses the front-end part to consume the web services developed in chapters 22 and 23. It covers jQuery Ajax method to call the web services APIs asynchronously. It also shows how to pass parameters via jQuery Ajax call.

Annexure B Possible Interview Questions and Answers: This annexure provides a list of interview questions along with the answers which you may face in your full stack development interview. You can use these questions to get an idea about the preparation you may need to do in order to clear the interview.

As part of this book, readers will also get **access to a Starter Project** which they can use as a blue print to develop the application of their choice. There are dedicated chapters in the book which shows the step-by-step process on creating this starter kit so readers know how to develop this type of project by themselves.

Icons Used in This Book

QUICK CHALLENGE

An out-of-the-box question that is not directly related to the technical topic and kind of a general knowledge question. It will induce stimulating thoughts in the readers' minds.



A question that will test your knowledge on the concept(s) being discussed.



This points out something important we would like readers' to take note of.

Instructor Resources

The following resources are available for instructors on request. To register, log onto https://www.wileyindia.com/Instructor_Manuals/Register/login.php

- 1.** Chapter-wise Solution Manuals.
- 2.** Chapter-wise PowerPoint Presentations (PPTs).

These Instructor Resources are presentation decks (one for each chapter) which can be taken to class directly or can be customized as per your requirements.

About the Author

Mayur Ramgir has more than 18 years of experience in the software industry, working at various levels. He is a Sun Certified Java Programmer and Oracle Certified SQL Database Expert. He has completed MS in computational science and engineering from Georgia Tech, USA, and M.Sc. in multimedia application and virtual environments from University of Sussex, UK. He has also attended in-class courses in various Universities like MIT Applied Software Security Certificate and University of Oxford System and Software Security Certificate.

Mayur Ramgir is an international award-winning innovator, author, serial-tech entrepreneur, speaker, and philanthropist. He is a fellow of The World Technology Network (WTN), which is a curated membership community comprised of the world's most innovative individuals and organizations in science, technology, and related fields. The WTN and its members – those creating the 21st century – are focused on exploring what is imminent, possible, and important in and around emerging technologies. Other members of WTN are Elon Musk – founder and CEO of SpaceX, co-founder and CEO of Tesla Inc., and co-founder of PayPal; Tim Berners-Lee – Founder of Internet WWW; Emmanuel Macron – President of France; John P. Holdren – Former Director of the White House Office of Science and Technology Policy (Obama administration); and Xi Jinping – President, People's Republic of China.



Mayur Ramgir is honored with the Champions of Change award by the Honorable Vice President of India, Mr. M. Venkaiah Naidu, on 26 December 2018, in the presence of former Chief Justice of India, K. G. Balakrishnan. He has been awarded the Pride of The Nation award by the Honorable Home Minister of India, Mr. Rajnath Singh, at the Vijay Diwas Celebration 2018 in New Delhi for his contribution in the field of innovation and philanthropy. He has been awarded Man of Excellence by Indian Achiever's Forum for his exceptional service for the betterment of society as change-maker.

Mayur Ramgir is the inventor of two patent pending technologies in USA. One of these is the ZPOD – MEDICAL KIOSK SYSTEM AND METHOD (USA patent pending 2017), is an IoT solution for medical field, disrupts the primary healthcare industry which will provide primary healthcare facilities to rural areas. Many developing countries like India are struggling to provide the primary health care facilities in many part of the country. This innovation is going to transform the life of millions by providing world-class doctors to the remotest part of the world.

Mayur Ramgir was featured on various TV and printed media including Fox News, NBC News, CBS News, Fox Business, Bloomberg International TV, Forbes, Inc. magazine, Daily Mirror, and The Huffington Post. He is also a contributing author of the *New York Daily Newspaper*, *Software Development Times* magazine, *Newsmax Finance*, and Singapore's top entrepreneurship magazine *Young Upstarts*.

To learn more about the author and to contact him, go to www.mayurramgir.com.

List of Video Content

Chapter	Section	Video Title	Video Content	Page No.
1	1.2 What is Full Stack Web Development?	Opportunities for Full Stack Developers	<ul style="list-style-type: none"> • Benefits of learning full stack development. • What are the opportunities after learning full stack development? • Career path for a full stack developer. 	1
	1.3 Introduction to Web Application Development	Importance of Web Application Development	<ul style="list-style-type: none"> • What is web application development? • Importance of designing to keep mobile in mind. • Essential elements to consider while developing for web. 	3
	1.8 Introduction to Web Services: API-Based Architecture with REST	Importance of Web Services	<ul style="list-style-type: none"> • What are the benefits of web services model? • Why to use REST over SOAP? • Difference between XML and JSON. 	12
2	2.3 What is E-Commerce?	E-Commerce – The Future	<ul style="list-style-type: none"> • What is e-commerce? Examples of e-commerce. • Benefits of e-commerce. • Things to consider for an e-commerce application. 	19
	2.5 Entity Relationship Diagram 2.6 UML Class Diagram 2.7 Flowchart 2.8 Front-End Page Flow Design	Important Elements for Application Architecture	<ul style="list-style-type: none"> • What is ERD? • What is UML? • What is flowchart? • What is page flow design? • Why should we consider these before designing our application? 	22 22 22 31
	2.9.3 GET versus POST	GET vs POST: Who Wins When?	<ul style="list-style-type: none"> • What is GET? • What is POST? • Which one to use and when. 	33
	3.1 Overview of HTML	Irreplaceable Part of Web Application Development	<ul style="list-style-type: none"> • What is HTML? • Why HTML? • What are the benefits of HTML? • How to use HTML. 	37
3	3.3 Important Components of HTML	What makes HTML?	<ul style="list-style-type: none"> • What are the important parts of HTML? • What is the difference between tags and elements? • Role of global attributes. • Benefits of event handlers. 	39
	3.11 Attributes to Style HTML Elements	Inline CSS	<ul style="list-style-type: none"> • What is inline CSS? • What are the benefits of using inline CSS? • Examples of inline CSS. 	66

Chapter	Section	Video Title	Video Content	Page No.
4	4.2 Overview of CSS 4.3 Relationship Between HTML and CSS	Beautifying the HTML Page	<ul style="list-style-type: none"> • What is CSS? • How to use CSS in HTML • How can CSS be used to beautify HTML pages? 	73 73
	4.17 Responsiveness	Make Page Presentable for Mobile with CSS	<ul style="list-style-type: none"> • What is Responsiveness? • Why should one care about Responsiveness? • CSS tries for Responsiveness. 	102
5	5.1 Overview of jQuery	World of jQuery	<ul style="list-style-type: none"> • The idea of jQuery. • jQuery vs JavaScript. • Benefits of jQuery. 	117
	5.4 Selectors	Importance of Selectors	<ul style="list-style-type: none"> • What are Selectors? • Use of Selectors. • Examples of Selectors. 	119
	5.6 Effects	Animate Your Web Application with CSS	<ul style="list-style-type: none"> • What are Effects? • How do Effects make application look cool? • Examples of Effects. 	126
6	6.1 Overview of Bootstrap	Make Your Application Look Good on All Devices	<ul style="list-style-type: none"> • What is Bootstrap? • Why should one use Bootstrap? • How does it work? 	177
	6.3 Grids	Making Tables Presentable for All Devices	<ul style="list-style-type: none"> • What is the need of Grids? • Table vs Grids. • Example of Grids. 	179
	6.8 Alerts	Notify Users With Beautiful On-Screen Messages	<ul style="list-style-type: none"> • What are Alerts? • Boring JavaScript Alert vs Bootstrap Alert. • Need for Bootstrap Alerts. 	191
	6.11 Progress Bars 6.12 Pagination 6.13 Cards 6.14 Navigation Menus 6.16 Forms 6.17 Carousel	Six Blessings of Bootstrap	<ul style="list-style-type: none"> • What is Progress Bar? What are its needs? • What is Pagination? What are its needs? • What is Card? How do we use it? • What is Navigation Menu? How is it different? • What are the benefits of Bootstrap Forms? • What is Carousel? Why do we need it? 	201 206 208 209 217 219
	6.18 Media Objects	Beautify User Comments	<ul style="list-style-type: none"> • What is Media Object? • How do we use it? 	223
7	7.3 Getting Started with HTML Pages	Build HTML Front-End	<ul style="list-style-type: none"> • How do we start creating HTML page? • What is the structure of the page? • Example of an HTML page. 	231
	7.4 Adding CSS to the HTML Page	Make the Pages Look Alive	<ul style="list-style-type: none"> • How to add CSS to HTML. • Examples of CSS. 	238

Chapter	Section	Video Title	Video Content	Page No.
8	8.1 Getting Started with jQuery 8.2 Home Page with jQuery	Use of jQuery	<ul style="list-style-type: none"> • How to use jQuery. • jQuery example on a page. 	243 243
9	9.1 Setting up Environment 9.1.1 Home Page with Bootstrap	Bootstrap Setup and Configuration	<ul style="list-style-type: none"> • How to get Bootstrap on a page. • Design a page with Bootstrap. 	251 253
10	10.1 Overview of Java	What is Java?	<ul style="list-style-type: none"> • What is Java? • Brief History of Java. • Java 13 updates. 	259
	10.2.1 Java Virtual Machine 10.2.2 Compiler 10.2.3 Java Runtime Environment 10.2.4 Java Development Kit	Core Elements of Java	<ul style="list-style-type: none"> • What is Java Virtual Machine? How does it work? • What is Compiler? • What is Java Runtime Environment? • What is Java Development Kit? 	263 263 263 264
	10.2.3 Java Runtime Environment 10.2.4 Java Development Kit	JDK vs JRE	<ul style="list-style-type: none"> • Difference between JDK and JRE. 	263 264
	10.2.5 Garbage Collector	Performance Enhancement with Garbage Collector	<ul style="list-style-type: none"> • What is Garbage collector? • How does Garbage collector work? • Principal of Garbage collection. • Performance improvements. 	265
	10.4 Programming in Java	Elements to Learn Before You Program in Java	<ul style="list-style-type: none"> • Install JDK. • Syntax. • Declaration Rules. • Identifiers. 	270
	10.4.6 Access Modifiers	Controlling Resource Access with Access Modifiers	<ul style="list-style-type: none"> • What are access modifiers? • Class Level vs Method Level. • Public vs Private vs Default vs Protected. 	274
	10.4.7 Non-Access Modifiers	Essential Non-Access Modifiers	<ul style="list-style-type: none"> • Final. • Abstract. 	276
	10.4.10 Classes and Objects	Class vs Objects	<ul style="list-style-type: none"> • What is Class? • What is Object? • Difference between Class and Object. 	278
	10.5.4.2 “super” keyword	World of Super	<ul style="list-style-type: none"> • What is Super? • How does it work? 	283
	10.6 New Features in Java 9	Exciting Features of Java 9	<ul style="list-style-type: none"> • Java Shell. • Multi-Release Java Files. • Java Linker. • Hash Algorithms. • Modular Java Packaging. • G1GC as default collector. 	287
	10.7 Eclipse IDE for Programming	Use of Eclipse for Writing Java Programs	<ul style="list-style-type: none"> • What is Eclipse? • Advantages of Eclipse. • Interface of Eclipse. • How do we use it? 	291

Chapter	Section	Video Title	Video Content	Page No.
11	11.1 Building Blocks of Java	What Makes Java?	<ul style="list-style-type: none"> • Keywords. • Primitive Classes. • Literals. • Variables. • Code Blocks. • Main Method. • Static vs Non-Static. 	299
	11.3 String Options	String Builder vs String Buffer	<ul style="list-style-type: none"> • What is String Builder? • String Builder vs String Buffer. 	304
	11.4 Arrays	World of Arrays	<ul style="list-style-type: none"> • What is Array? • How does it work? • Example of Array. 	305
	11.5 Enums	Enums Wonder	<ul style="list-style-type: none"> • What is Enum? • Example of Enum. 	307
	11.6 Wrapper Classes	Doorway for Primitives to the Object World	<ul style="list-style-type: none"> • What are wrapper classes? • Primitive and their equivalent wrapper classes. 	309
	11.7 Autoboxing and Unboxing	Autoboxing vs Unboxing	<ul style="list-style-type: none"> • What is Autoboxing? • What is Unboxing? • Autoboxing vs Unboxing. 	311
	11.8.1 Various Types of Operators	Various Types of Operators	<ul style="list-style-type: none"> • Arithmetic Operators. • Unary Operators. • Assignment Operators. • Logical Operators. • Ternary Operators. • Relational Operators. • Shift Operators. • Bitwise Operators. 	313
	11.9 Control Flow	Control Flow	<ul style="list-style-type: none"> • Conditional Statements. • Iteration Statements. • Jump Statements. 	327
12	12.2 Object-Oriented Programming Principles	Object Oriented Programming Principles	<ul style="list-style-type: none"> • What is Object Oriented Programming? • What is Inheritance? • What is Polymorphism? • What is Abstraction? • What is Encapsulation? 	338
	12.5 Overriding and Overloading	Overriding vs Overloading	<ul style="list-style-type: none"> • What is Overriding? • What is Overloading? • Overriding vs Overloading. 	353
	12.6 Coupling and Cohesion	Coupling and Cohesion	<ul style="list-style-type: none"> • What is coupling? • What is cohesion? 	356

Chapter	Section	Video Title	Video Content	Page No.
13	13.2 Generic Programming	World of Generics	<ul style="list-style-type: none"> • What is Generic Programming? • Generic Methods. • Generic Programming Use. 	363
	13.2.4 Overriding <code>toString()</code> , <code>hashCode()</code> , and <code>equals()</code>	Overriding <code>toString()</code> , <code>hashCode()</code> , and <code>equals()</code>	<ul style="list-style-type: none"> • What is <code>toString()</code>? • What is <code>hashCode()</code>? • What is <code>equal()</code>? • Why should we override them? 	367
	13.3 Collections	World of Collections	<ul style="list-style-type: none"> • What is Collection? • Benefits of Java Collections. • List of major Collection Interfaces. • <code>ArrayList</code> vs <code>Vector</code>. • <code>Comparable</code> vs <code>Comparator</code>. 	373
14	14.2 Understanding Error Handling	Understanding Error Handling	<ul style="list-style-type: none"> • What is error handling? • Difference between Exception and Error. 	403
	14.3 Logical Errors	Logical Errors	<ul style="list-style-type: none"> • What are the logical errors? 	404
	14.4 Syntactical Errors	Syntactical Errors	<ul style="list-style-type: none"> • What are syntactical errors? • Capitalization, Splitting Strings, Not importing classes. 	406
	14.5 Semantic Errors	Semantic Errors	<ul style="list-style-type: none"> • What are the semantic errors? • Improper use of operators. • Incompatible types. • Precision. • Scoping. 	408
15	15.2 Garbage Collection in Java	World of Garbage Collection	<ul style="list-style-type: none"> • What is Garbage Collection? • How does Garbage Collection work? 	415
	15.4 G1 and CMS Garbage Collectors	Types of Garbage Collectors	<ul style="list-style-type: none"> • What is G1 Garbage Collector? • What is CMS Garbage Collector? 	417
	15.5 Advantages of Garbage Collection in Java	Advantages of Garbage Collection	<ul style="list-style-type: none"> • What are the advantages of Garbage Collection? 	418
	15.6 Making Objects Eligible for Garbage Collection	Making Objects Eligible for Garbage Collection	<ul style="list-style-type: none"> • What are unreachable objects? • How do you reassign reference variables? • How to nullified reference variables. • What are Anonymous Objects? 	418
	15.7 JEP 318 – Epsilon: A No-Op Garbage Collector	No-Op Garbage Collector	<ul style="list-style-type: none"> • What is No-Op Garbage Collector? 	422
16	16.2 Role of Strings in Java	Role of Strings	<ul style="list-style-type: none"> • What is the role of string in Java? • Various String functions. 	425
	16.3 Types of String Operations	Types of String Operations	<ul style="list-style-type: none"> • How to use Concatenation. • How to split string. 	428
	16.4 StringBuillder and StringBuffer Explained	StringBuillder and StringBuffer	<ul style="list-style-type: none"> • What is <code>StringBuilder</code>? • What is <code>StringBuffer</code>? 	433
	16.5 Java I/O	Input Output Operations	<ul style="list-style-type: none"> • What is <code>InputStream</code>? • What is <code>OutputStram</code>? • What are the methods of <code>OutputStream</code>? • What are the methods of <code>InputStream</code>? 	436
	16.6 File Management in Java	Java File Management	<ul style="list-style-type: none"> • How file management works in Java. • <code>FileReader</code> and <code>FileWriter</code>. 	438

Chapter	Section	Video Title	Video Content	Page No.
17	17.2 Introduction to Data Structures	Data Structures	<ul style="list-style-type: none"> • What is data structure? 	441
	17.3 Classification of Data Structures	Classification of Data Structure	<ul style="list-style-type: none"> • How are data structures classified? • What are Primitive Data structures? • What are Non-primitive Data structures? • Tree data structure. • Types of tree data structure. 	442
18	18.2 Functional Programming	Fundamentals of Functional Programming	<ul style="list-style-type: none"> • What is functional programming? • What is the benefit of functional programming? • What are the fundamental concepts of functional programming? • What is Pure Function? • What is Recursion? • What is Evaluation? 	459
	18.4 Object-Oriented versus Functional Programming	World of Lambdas	<ul style="list-style-type: none"> • What is Lambda? • How to use Lambda. • Various elements of Lambda expression. 	464
19	19.2 Reactive Programming	Reactive Programming World	<ul style="list-style-type: none"> • What is reactive programming? • Examples of reactive programming. • Benefits of reactive programming. 	485
	19.4 What is Multithreading?	World of Multithreading	<ul style="list-style-type: none"> • What is Multithreading? • What is the need of Multithreading? • How to use Multithreading in Java. 	488
	19.5 Concurrency	World of Concurrency	<ul style="list-style-type: none"> • What is concurrency? • How does concurrency work? • Concurrency API improvements. • Synchronizing Code Blocks. • Understanding Deadlock. 	494
	19.7 Concurrent Data Structures	World of Concurrent Data Structure	<ul style="list-style-type: none"> • What is concurrent data structure? • What is the need of concurrent data structure? • How to use concurrent data structure. 	506
	19.9 Designing Concurrent Java Programs	Concurrent Java Programs	<ul style="list-style-type: none"> • How to design concurrent Java programs. • Scalability option. • High-Level Concurrency. • Immutable objects. • Lazy initialization. 	511
20	20.1 Spring Framework	World of Spring	<ul style="list-style-type: none"> • What is Spring? • What is Spring Framework? • How does Spring work? • What are the elements of Spring? • What is Inversion of Control? • What is Bean Scope? • What is Dependency Injection? 	517

Chapter	Section	Video Title	Video Content	Page No.
20	20.2 Spring Architecture	Spring Architecture	<ul style="list-style-type: none"> • What is Spring Architecture? • What are the core containers? • What is Data Access/Integration? • What is Web Container? 	525
	20.3 Spring MVC	World of Model View Controller	<ul style="list-style-type: none"> • What is Spring MVC? • What is Dispatcher Servlet? • What is Context Hierarchy? • How does Spring MVC work? 	527
	20.4 Interception	Interception in Spring MVC	<ul style="list-style-type: none"> • What is Interception? 	528
	20.5 Chain of Resolvers	Chain of Resolvers in Spring MVC	<ul style="list-style-type: none"> • What is Chain of Resolvers? • How does Chain of Resolvers work? 	528
	20.6 View Resolution	View Resolution in Spring MVC	<ul style="list-style-type: none"> • How does view resolution work in Spring MVC? 	529
	20.9 Model Interface	Model Interface in Spring MVC	<ul style="list-style-type: none"> • What is model interface? 	535
21	21.2 Architecture	Hibernate Architecture	<ul style="list-style-type: none"> • What is Hibernate Architecture? • What is configuration object? • What is session? • What is method? • What is cache? 	566
	21.3 Installation and Configuration	Installation and Configuration of Hibernate	<ul style="list-style-type: none"> • How to install Hibernate. • How to configure Hibernate in Spring application. • How to setup database. 	568
	21.5 Inheritance Mapping	Inheritance Mapping in Hibernate	<ul style="list-style-type: none"> • What is inheritance mapping in Hibernate? • How does Table per Hierarchy work? • How does Table per Subclass work? 	572
	21.6 Collection Mapping	Collection Mapping in Hibernate	<ul style="list-style-type: none"> • What is collection mapping? • What is collection mapping list? • What are the key elements? • How to map with list. • How to map with set. • How to map with map. 	579
	21.8 Hibernate Query Language	Hibernate Query Language	<ul style="list-style-type: none"> • What is Hibernate Query language? • How does Hibernate Query Language work? • What is Hibernate Named Query? 	588
	21.9 Caching	Caching in Hibernate	<ul style="list-style-type: none"> • What is caching? • How does it work in Hibernate? 	592
	21.10 Spring Integration	Spring Integration	<ul style="list-style-type: none"> • How to integrate with Spring framework. 	594

Chapter	Section	Video Title	Video Content	Page No.
22	22.1 Setting up Environment	Spring Hibernate Development Environment Setup	<ul style="list-style-type: none"> • How to setup development environment. • What is Spring Tool Suite? • How to install Spring Tool Suite? • Spring Tool Suite perspectives. 	599
	22.2 Creating a New Project	Creating a Spring MVC Webservice with Hibernate Project	<ul style="list-style-type: none"> • How to create a new Spring MVC project. 	602
	22.3 Creating Models	Creating Models in Spring MVC	<ul style="list-style-type: none"> • How to create a Model in Spring MVC. 	611
	22.4 Creating Data Access Object	Creating Data Access Object	<ul style="list-style-type: none"> • How to create data access object in Spring MVC. 	618
	22.5 Creating Controller	Creating Controller in Spring MVC	<ul style="list-style-type: none"> • How to create controller in Spring MVC. 	628
23	23.1 Installing MySQL	Installing MySQL	<ul style="list-style-type: none"> • How to install MySQL in Windows. • How to install MySQL in Linux. • How to install MySQL in Mac. 	637
	23.2 Create Database and Tables	Create Database and Tables	<ul style="list-style-type: none"> • How to create database in MySQL. • How to create tables in MySQL. 	638
	23.3 Making DAO to Perform CRUD	How to Perform CRUD Operation Using DAO	<ul style="list-style-type: none"> • How to perform CURD operation using DAO. 	646
Annexure A	A.1 Endpoints Connections	Web API Calls Using jQuery	<ul style="list-style-type: none"> • How to call web APIs from jQuery. • How to use Ajax call. 	651

Contents

<i>Preface</i>	vii
<i>About the Author</i>	xi
<i>List of Video Content</i>	xiii
Chapter 1 Introduction to Full Stack Development	1
1.1 Introduction	1
1.2 What is Full Stack Web Development?	1
1.2.1 Front-end	2
1.2.2 Back-end	2
1.2.3 Technologies Essential for Full Stack Development	2
1.3 Introduction to Web Application Development	3
1.4 Front-End Technologies	3
1.4.1 HTML/CSS	4
1.4.2 JavaScript	6
1.4.3 jQuery	7
1.4.4 Bootstrap	8
1.5 Back-End Technologies (Server-Side)	11
1.6 Introduction to Back-end Development with Java	11
1.6.1 Java	11
1.6.2 Spring Framework	12
1.7 Introduction to Model View Controller (MVC)	12
1.8 Introduction to Web Services: API-Based Architecture with REST	12
1.8.1 REST	13
1.8.2 Simple Object Access Protocol (SOAP)	13
1.9 Communication Between Front-End and Back-End	14
1.9.1 JavaScript Object Notation (JSON)	14
1.9.2 Extensible Markup Language (XML)	15
1.9.3 Database	15
1.10 Introduction to Object Relational Mapping (ORM) with Hibernate	15
1.10.1 Hibernate	16
Summary	16
Multiple-Choice Questions	17
Review Questions	17
Exercises	17
Project Idea	17
Recommended Readings	18
Chapter 2 Getting Started with Full Stack Development: A Project Idea	19
2.1 Introduction	19
2.2 Project Outline	19
2.3 What is E-Commerce?	19
2.3.1 Advantages of E-Commerce	19
2.3.2 Important Considerations while Developing an E-Commerce Website	20
2.4 Required Entities	21
2.5 Entity Relationship Diagram	22
2.6 UML Class Diagram	22

2.7	Flowchart	22
2.7.1	Login–Registration–Ordering Flowchart	25
2.7.2	Cart Finalization Flowchart	26
2.7.3	Order Processing Flowchart	27
2.7.4	Vendor Order Processing Flowchart	28
2.7.5	Admin Dashboard Flowchart	29
2.8	Front-End Page Flow Design	31
2.9	Back-End Web Services API Endpoints	32
2.9.1	GET Endpoints	32
2.9.2	POST Endpoints	32
2.9.3	GET versus POST	33
	<i>Summary</i>	34
	<i>Multiple-Choice Questions</i>	34
	<i>Review Questions</i>	34
	<i>Exercises</i>	35
	<i>Project Idea</i>	35
	<i>Recommended Readings</i>	35
Chapter 3	Introduction to Hyper Text Markup Language	37
3.1	Overview of HTML	37
3.2	Getting Started with HTML Code	38
3.3	Important Components of HTML	39
3.3.1	Tags	39
3.3.2	Elements	39
3.3.3	Attributes	39
3.3.4	Global Attributes	40
3.3.5	Event Handler Content Attributes	42
3.3.6	Headings	45
3.3.7	<head> Tag	47
3.3.8	Paragraphs	49
3.3.9	Line Breaks	49
3.4	Text Formatting Tags	50
3.4.1	Bold Text	50
3.4.2	Strong	50
3.4.3	Italic Text	51
3.4.4	Emphasized Text	51
3.4.5	Small Text	52
3.4.6	Marked Text	52
3.4.7	Removed Text	53
3.4.8	Inserted Text	53
3.4.9	Subscripted Text	54
3.4.10	Superscripted Text	54
3.5	Quotations	54
3.5.1	Short Quotations	54
3.5.2	Quotation from Other Sources	55
3.5.3	Abbreviations	56
3.5.4	Contact Information	56
3.6	Comments	57
3.7	Links	57
3.7.1	The Target Attribute	58
3.7.2	Images as Links	58
3.7.3	Bookmark	59

3.8	Images	61
3.8.1	Alt Attribute	61
3.9	Tables	62
3.10	Lists	64
3.11	Attributes to Style HTML Elements	66
3.11.1	Style Attribute	66
3.11.2	Text Color	67
3.11.3	Fonts	67
3.11.4	Size of Text	68
3.11.5	Text Alignment	68
3.11.6	Different Screen Sizes	68
3.11.7	No Closing Tag	69
	<i>Summary</i>	69
	<i>Multiple-Choice Questions</i>	70
	<i>Review Questions</i>	70
	<i>Exercises</i>	70
	<i>Project Idea</i>	70
	<i>Recommended Readings</i>	71

Chapter 4 Introduction to Cascading Style Sheets 73

4.1	Introduction	73
4.1.1	History	73
4.2	Overview of CSS	73
4.3	Relationship Between HTML and CSS	73
4.4	How Does CSS Work?	74
4.4.1	Advantages of Using CSS	74
4.4.2	HTML and Styling	74
4.5	Syntax	74
4.5.1	The “ <i>id</i> ” Selector	75
4.5.2	The “ <i>class</i> ” Selector	76
4.5.3	Groups of Selectors	77
4.5.4	Comments	77
4.6	Different Methods to Integrate CSS with HTML	78
4.6.1	External Style Sheet	78
4.6.2	Internal Style Sheet	79
4.6.3	Inline Style	79
4.7	Colors	80
4.7.1	Color Names	80
4.7.2	Color Values	81
4.8	Backgrounds in CSS	82
4.9	Setting up Height and Width of an Element	85
4.10	Box Model	87
4.11	CSS Outline	88
4.12	Text in CSS	91
4.13	Fonts	94
4.14	Links in CSS	95
4.15	Lists in CSS	98
4.16	Tables in CSS	100
4.17	Responsiveness	102
4.18	Position Property in CSS	104
4.19	Navigation Bars	108

4.20	Dropdown	111
4.21	Forms	113
	<i>Summary</i>	114
	<i>Multiple-Choice Questions</i>	115
	<i>Review Questions</i>	115
	<i>Exercises</i>	115
	<i>Project Idea</i>	116
	<i>Recommended Readings</i>	116
Chapter 5 Introduction to jQuery		117
5.1	Overview of jQuery	117
5.2	Configuration of jQuery	117
5.3	Syntax	118
5.4	Selectors	119
	5.4.1 Element	119
	5.4.2 #id Selector	120
	5.4.3 Class Selector	121
5.5	Events	122
	5.5.1 Syntax	122
	5.5.2 jQuery Event Methods	122
5.6	Effects	126
	5.6.1 hide() and show()	126
	5.6.2 toggle()	128
	5.6.3 Fading	128
	5.6.4 Sliding	130
	5.6.5 Animation	132
	5.6.6 jQuery Stop Animations	135
	5.6.7 Callback Function	136
	5.6.8 Chaining	137
5.7	Working with HTML	138
	5.7.1 DOM Manipulation	138
	5.7.2 Get Content and Attributes	138
	5.7.3 Set Content and Attributes	142
	5.7.4 Addition of Elements	144
	5.7.5 Deleting HTML Elements	149
5.8	jQuery with CSS	153
	5.8.1 addClass()	153
	5.8.2 removeClass()	155
	5.8.3 toggleClass()	156
	5.8.4 css()	156
5.9	Traversing	158
	5.9.1 Ancestors	158
	5.9.2 Descendants	162
	5.9.3 Siblings	164
	5.9.4 Filtering	171
	<i>Summary</i>	174
	<i>Multiple-Choice Questions</i>	174
	<i>Review Questions</i>	175
	<i>Exercises</i>	175
	<i>Project Idea</i>	175
	<i>Recommended Readings</i>	175

Chapter 6	Introduction to Bootstrap	177
6.1	Overview of Bootstrap	177
6.1.1	Prerequisites	177
6.1.2	Installation	177
6.2	Structure of a Bootstrap-enabled Webpage	178
6.2.1	Container Example	179
6.3	Grids	179
6.3.1	Classes	179
6.4	Typography	182
6.4.1	Display Headings	182
6.4.2	HTML Text Elements in Bootstrap	183
6.5	Colors	185
6.5.1	Color Classes to Convey Meaning	185
6.5.2	Background Colors	186
6.6	Images	187
6.6.1	Rounded Images	187
6.6.2	Circle Images	188
6.6.3	Thumbnail Image	189
6.7	Jumbotron	190
6.8	Alerts	191
6.8.1	Closing Alerts	194
6.8.2	Animated Alerts	195
6.9	Buttons	196
6.9.1	Outline and Size	197
6.10	Button Groups	198
6.10.1	Vertical	200
6.10.2	Dropdown Menus	200
6.11	Progress Bars	201
6.11.1	Height and Label	202
6.11.2	Adding Colors to Progress Bar	203
6.11.3	Adding Stripes to Progress Bars	204
6.11.4	Adding Animations to the Progress Bar	205
6.12	Pagination	206
6.13	Cards	208
6.14	Navigation Menus	209
6.14.1	Vertical Nav Menus	210
6.14.2	Tabs	212
6.14.3	Pills	212
6.15	Navigation Bar	213
6.15.1	Colorful Navigation Bars	214
6.16	Forms	217
6.16.1	Stacked Form	218
6.16.2	Bootstrap Inline Form	218
6.17	Carousel	219
6.18	Media Objects	223
	<i>Summary</i>	227
	<i>Multiple-Choice Questions</i>	228
	<i>Review Questions</i>	228
	<i>Exercises</i>	228
	<i>Project Idea</i>	228
	<i>Recommended Readings</i>	229

Chapter 7	Build Pages for MyEShop with HTML and CSS	231
7.1	Setting up Environment	231
7.2	Identify the Pages	231
7.3	Getting Started with HTML Pages	231
7.3.1	Home Page	233
7.3.2	Header	234
7.3.3	Table	234
7.3.4	Logo	234
7.3.5	Search Bar	235
7.3.6	Category List	235
7.3.7	Search Button	235
7.3.8	Shopping Cart	235
7.3.9	Navigation Menu – Home	236
7.3.10	Execution of Code	236
7.4	Adding CSS to the HTML Page	238
7.4.1	Background Color	238
7.4.2	Heading: Center Align	238
7.4.3	Navigation Menu Color	239
7.4.4	Fix the Image Problem	239
	<i>Summary</i>	240
	<i>Multiple-Choice Questions</i>	241
	<i>Review Questions</i>	241
	<i>Exercises</i>	241
	<i>Project Idea</i>	241
	<i>Recommended Readings</i>	241
Chapter 8	Use of jQuery on HTML CSS	243
8.1	Getting Started with jQuery	243
8.2	Home Page with jQuery	243
	<i>Summary</i>	248
	<i>Multiple-Choice Questions</i>	248
	<i>Review Questions</i>	248
	<i>Exercises</i>	248
	<i>Project Idea</i>	248
	<i>Recommended Readings</i>	249
Chapter 9	Use of Bootstrap to Make HTML Responsive	251
9.1	Setting up Environment	251
9.1.1	Home Page with Bootstrap	253
	<i>Summary</i>	257
	<i>Multiple-Choice Questions</i>	257
	<i>Review Questions</i>	257
	<i>Exercises</i>	257
	<i>Project Idea</i>	257
	<i>Recommended Readings</i>	258
Chapter 10	Introduction to Java Language	259
10.1	Overview of Java	259
10.2	Basic Java Concepts	262

10.2.1 Java Virtual Machine	263
10.2.2 Compiler	263
10.2.3 Java Runtime Environment	263
10.2.4 Java Development Kit	264
10.2.5 Garbage Collector	265
10.2.6 Objects in Java	265
10.3 Principles of Object-Oriented Programming in Java	266
10.3.1 IS-A and HAS-A Relationships	266
10.3.2 Abstraction	267
10.3.3 Encapsulation	268
10.3.4 Inheritance	269
10.3.5 Polymorphism	269
10.4 Programming in Java	270
10.4.1 Installing a JDK	271
10.4.2 Using Libraries	271
10.4.3 Syntax	271
10.4.4 Declaration Rules	273
10.4.5 Identifiers	273
10.4.6 Access Modifiers	274
10.4.7 Non-Access Modifiers	276
10.4.8 Reserved Words	277
10.4.9 The Keyword “this”	277
10.4.10 Classes and Objects	278
10.5 Java Packages	279
10.5.1 Structure	279
10.5.2 Variables	280
10.5.3 Methods	281
10.5.4 Constructors	281
10.5.5 Other Methods	285
10.5.6 Programming Examples	285
10.6 New Features in Java 9	287
10.6.1 Java Shell	287
10.6.2 Multi-Release JAR Files	287
10.6.3 Platform Support for Legacy Versions	287
10.6.4 Java Linker	288
10.6.5 Hash Algorithms	288
10.6.6 Modular Java Packaging	288
10.6.7 Tuning Improvements	290
10.6.8 G1GC as Default Collector	290
10.6.9 Process API Updates	291
10.6.10 XML Catalog	291
10.7 Eclipse IDE for Programming	291
10.7.1 Advantages of Using Eclipse	292
10.7.2 Setting Up Eclipse	292
<i>Summary</i>	297
<i>Multiple-Choice Questions</i>	297
<i>Review Questions</i>	298
<i>Exercises</i>	298
<i>Project Idea</i>	298
<i>Recommended Readings</i>	298

Chapter 11	Language Syntax and Elements of Language	299
11.1	Building Blocks of Java	299
11.1.1	Keywords	300
11.1.2	Primitive Classes	301
11.1.3	Literals	302
11.1.4	Variables	302
11.1.5	Code Blocks	302
11.1.6	Comments	303
11.2	Calling the Main Method	303
11.2.1	Static and Non-Static Methods	303
11.3	String Options	304
11.4	Arrays	305
11.5	Enums	307
11.6	Wrapper Classes	309
11.6.1	Outline of Wrapper Classes	309
11.6.2	Creation of Wrapper Objects	309
11.6.3	Wrapper Conversion Utility Methods	310
11.7	Autoboxing and Unboxing	311
11.8	Developing Logic	313
11.8.1	Various Types of Operators	313
11.8.2	Expressions	326
11.9	Control Flow	327
11.10	Loops	329
11.10.1	While Loop	329
11.10.2	For Loop	330
11.11	Branching	331
	<i>Summary</i>	334
	<i>Multiple-Choice Questions</i>	334
	<i>Review Questions</i>	334
	<i>Exercises</i>	334
	<i>Project Idea</i>	335
	<i>Recommended Readings</i>	335
Chapter 12	Object-Oriented Programming	337
12.1	Introduction	337
12.1.1	History of Object-Oriented Programming	337
12.2	Object-Oriented Programming Principles	338
12.2.1	Encapsulation	339
12.2.2	Abstraction	341
12.2.3	Inheritance	343
12.2.4	Polymorphism	345
12.3	Object-Oriented Programming Principles in Application	350
12.3.1	More About Objects	350
12.3.2	Classes and Object-Oriented Programming Principles	351
12.4	Understanding an Interface	351
12.5	Overriding and Overloading	353
12.5.1	Overriding	353
12.5.2	Overloading	354
12.6	Coupling and Cohesion	356
12.6.1	Coupling	357
12.6.2	Cohesion	357
12.7	Implementation in Java	357

12.7.1 Objects Work as Solutions	357
12.7.2 Ideal Tips for Implementing Basic OOP Concepts	358
12.8 Future of Object-Oriented Programming	359
12.8.1 The Downfalls	359
12.8.2 Future Applications	359
12.9 Understanding the World	360
<i>Summary</i>	360
<i>Multiple-Choice Questions</i>	361
<i>Review Questions</i>	361
<i>Exercises</i>	361
<i>Project Idea</i>	362
<i>Recommended Readings</i>	362
Chapter 13 Generics and Collections	363
13.1 Introduction	363
13.2 Generic Programming	363
13.2.1 Benefits of Generics	364
13.2.2 Using Generics in Java	365
13.2.3 Generic Methods	366
13.2.4 Overriding <code>toString()</code> , <code>hashCode()</code> , and <code>equals()</code>	367
13.3 Collections	373
13.3.1 Collections in Java	374
13.3.2 Benefits of Java Collections	374
13.3.3 Collection Interfaces	375
13.3.4 Collections Classes	377
13.4 Implementing Collection Classes	378
13.4.1 Map Interface	378
13.4.2 Set Interface	383
13.4.3 List Interface	386
13.4.4 Queue Interface	390
13.4.5 Stream API	392
13.5 List of Key Methods for Arrays and Collections	393
13.5.1 Arrays (<code>java.util.Arrays</code>)	393
13.5.2 Collections (<code>java.util.Collections</code>)	394
13.5.3 Key Methods for List, Set, Map, and Queue	394
<i>Summary</i>	400
<i>Multiple-Choice Questions</i>	401
<i>Review Questions</i>	401
<i>Exercises</i>	401
<i>Project Idea</i>	401
<i>Recommended Readings</i>	401
Chapter 14 Error Handling	403
14.1 Introduction	403
14.2 Understanding Error Handling	403
14.3 Logical Errors	404
14.4 Syntactical Errors	406
14.4.1 Capitalization	406
14.4.2 Splitting Strings	406
14.4.3 Not Importing Classes	407

14.4.4	Different Methods	407
14.4.5	Curly Braces	407
14.5	Semantic Errors	408
14.5.1	Improper Use of Operators	408
14.5.2	Incompatible Types	408
14.5.3	Precision	408
14.5.4	Scoping	408
14.6	Importance of Error Handling	409
14.6.1	Try, Catch, and Finally	410
14.7	Checked versus Runtime Exceptions	412
14.7.1	Checked Exceptions	412
14.7.2	Runtime Exceptions	413
	<i>Summary</i>	413
	<i>Multiple-Choice Questions</i>	414
	<i>Review Questions</i>	414
	<i>Exercises</i>	414
	<i>Project Idea</i>	414
	<i>Recommended Readings</i>	414
Chapter 15	Garbage Collection	415
15.1	Introduction	415
15.2	Garbage Collection in Java	415
15.3	Major Garbage Collection	416
15.4	G1 and CMS Garbage Collectors	417
15.5	Advantages of Garbage Collection in Java	418
15.6	Making Objects Eligible for Garbage Collection	418
15.6.1	Unreachable Objects	418
15.6.2	Reassigning Reference Variables	420
15.6.3	Nullified Reference Variables	420
15.6.4	Anonymous Objects	421
15.7	JEP 318 – Epsilon: A No-Op Garbage Collector	422
	<i>Summary</i>	423
	<i>Multiple-Choice Questions</i>	423
	<i>Review Questions</i>	424
	<i>Exercises</i>	424
	<i>Project Idea</i>	424
	<i>Recommended Readings</i>	424
Chapter 16	Strings, I/O Operations, and File Management	425
16.1	Introduction	425
16.2	Role of Strings in Java	425
16.3	Types of String Operations	428
16.3.1	Concatenation	429
16.3.2	Splitting Strings	430
16.4	StringBuilder and StringBuffer Explained	433
16.5	Java I/O	436
16.5.1	InputStream	437
16.5.2	OutputStream	438
16.5.3	Methods of OutputStream	438
16.5.4	Methods of InputStream	438

16.6	File Management in Java	438
	<i>Summary</i>	439
	<i>Multiple-Choice Questions</i>	439
	<i>Review Questions</i>	440
	<i>Exercises</i>	440
	<i>Project Idea</i>	440
	<i>Recommended Readings</i>	440
Chapter 17	Data Structure and Integration in Program	441
17.1	Introduction	441
17.2	Introduction to Data Structures	441
17.3	Classification of Data Structures	442
	17.3.1 Primitive Data Structures	442
	17.3.2 Non-Primitive Data Structures	444
	<i>Summary</i>	457
	<i>Multiple-Choice Questions</i>	457
	<i>Review Questions</i>	457
	<i>Exercises</i>	458
	<i>Project Idea</i>	458
	<i>Recommended Readings</i>	458
Chapter 18	Lambdas and Functional Programming	459
18.1	Introduction	459
18.2	Functional Programming	459
	18.2.1 Fundamental Concepts of Functional Programming	460
18.3	Functional Programming in Java	462
	18.3.1 Pure Function	462
	18.3.2 Higher-Order Function	463
	18.3.3 No State	463
	18.3.4 Functional Interfaces	464
18.4	Object-Oriented versus Functional Programming	464
18.5	Lambdas	465
	18.5.1 Elements of Lambda Expression	465
	18.5.2 Examples	468
	18.5.3 Using return	470
	18.5.4 Lambdas with Loops	473
	18.5.5 Lambdas with Threads	473
18.6	Date and Time API	474
	18.6.1 Problems with Earlier APIs	474
	18.6.2 Local Date	475
	18.6.3 LocalTime	477
	18.6.4 LocalDateTime	479
	18.6.5 ZonedDateTime API	480
	<i>Summary</i>	482
	<i>Multiple-Choice Questions</i>	482
	<i>Review Questions</i>	482
	<i>Exercises</i>	482
	<i>Project Idea</i>	483
	<i>Recommended Readings</i>	483

Chapter 19 Multithreading and Reactive Programming	485
19.1 Introduction	485
19.2 Reactive Programming	485
19.3 Reactive Programming	487
19.4 What is Multithreading?	488
19.4.1 Multithreading in Java	489
19.4.2 Programming with Multithreading	490
19.5 Concurrency	494
19.5.1 Advantages of Concurrency	494
19.5.2 Concurrency in Java	494
19.5.3 Concurrency Support	497
19.5.4 Concurrency API Improvements	498
19.5.5 Dealing with Individual Threads	501
19.5.6 Synchronizing Code Blocks	502
19.6 Understanding Deadlock	504
19.6.1 Resolving Deadlock	506
19.7 Concurrent Data Structures	506
19.8 Multithreading Examples	506
19.8.1 Matrix Multiplication	508
19.9 Designing Concurrent Java Programs	511
19.9.1 Ideal Tips for Concurrent Java Programs	512
19.9.2 High-Level Concurrency	513
19.9.3 Controlling Sequential Executions	514
19.9.4 Avoiding Lazy initialization	514
<i>Summary</i>	514
<i>Multiple-Choice Questions</i>	515
<i>Review Questions</i>	515
<i>Exercises</i>	515
<i>Project Idea</i>	516
<i>Recommended Readings</i>	516
Chapter 20 Introduction to Spring and Spring MVC	517
20.1 Spring Framework	517
20.1.1 Inversion of Control	517
20.1.2 Dependency Injection	518
20.1.3 Aspect-Oriented Programming	523
20.2 Spring Architecture	525
20.2.1 Core Container	525
20.2.2 Data Access/Integration	526
20.2.3 Web Container	526
20.2.4 Other Modules	527
20.3 Spring MVC	527
20.3.1 DispatcherServlet	527
20.3.2 Spring MVC Processing	528
20.4 Interception	528
20.5 Chain of Resolvers	528
20.6 View Resolution	529
20.7 Multiple View Pages	531
20.8 Multiple Controllers	533
20.9 Model Interface	535

20.10	RequestParam	537
20.11	Form Tag Library	539
20.12	Form Text Field	539
20.13	CRUD Example	543
20.14	File Upload in Spring MVC	549
20.15	Validation in Spring MVC	551
20.16	Validation with Regular Expression	555
20.17	Validation with Numbers	558
	<i>Summary</i>	561
	<i>Multiple-Choice Questions</i>	562
	<i>Review Questions</i>	562
	<i>Exercises</i>	562
	<i>Project Idea</i>	562
	<i>Recommended Readings</i>	563

Chapter 21	Introduction to Hibernate	565
21.1	Introduction	565
21.2	Architecture	566
	21.2.1 Configuration Object	567
	21.2.2 Session	567
	21.2.3 Methods	567
	21.2.4 Hibernate Criteria	567
	21.2.5 Cache	568
21.3	Installation and Configuration	568
	21.3.1 Database	568
	21.3.2 Eclipse	568
	21.3.3 JDBC	568
	21.3.4 Annotations	569
21.4	Java Objects in Hibernate	571
	21.4.1 Primary Key	571
21.5	Inheritance Mapping	572
	21.5.1 Table Per Hierarchy	572
	21.5.2 Table Per Concrete Class	574
	21.5.3 Table Per Subclass	576
21.6	Collection Mapping	579
	21.6.1 Mapping List	579
	21.6.2 Key Element	580
	21.6.3 Collection Mapping with Lists	580
	21.6.4 Mapping with Set	582
21.7	Mapping with Map	583
	21.7.1 One-to-Many	586
	21.7.2 Transaction	588
21.8	Hibernate Query Language	588
	21.8.1 HCQL	590
	21.8.2 Hibernate Named Query	590
21.9	Caching	592
21.10	Spring Integration	594
	<i>Summary</i>	597
	<i>Multiple-Choice Questions</i>	597
	<i>Review Questions</i>	598

<i>Exercises</i>	598
<i>Project Idea</i>	598
<i>Recommended Readings</i>	598
Chapter 22 Develop Web Services for the APIs	599
22.1 Setting up Environment	599
22.2 Creating a New Project	602
22.3 Creating Models	611
22.4 Creating Data Access Object	618
22.5 Creating Controller	628
<i>Summary</i>	634
<i>Multiple-Choice Questions</i>	634
<i>Review Questions</i>	635
<i>Exercises</i>	635
<i>Project Idea</i>	635
<i>Recommended Readings</i>	635
Chapter 23 Develop Models with Hibernate	637
23.1 Installing MySQL	637
23.1.1 Windows	637
23.1.2 Windows/Mac	637
23.2 Create Database and Tables	638
23.2.1 Linking Tables to Models	639
23.2.2 Setting up Relationships	643
23.3 Making DAO to Perform CRUD	646
<i>Summary</i>	649
<i>Multiple-Choice Questions</i>	650
<i>Review Questions</i>	650
<i>Exercises</i>	650
<i>Project Idea</i>	650
<i>Recommended Readings</i>	650
Annexure A: Consuming Web Services	651
Annexure B: Possible Interview Questions and Answers	653
Annexure C: Answers to Objective Type Questions	667
Index	671

Introduction to Full Stack Development

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Concept of full stack development.
- Essential technologies of full stack development.
- Web application development.
- Model–View–Controller with JSON, XML, etc.
- Object-relational mapping achieved through Hibernate.
- Front-end development with web technologies such as HTML, jQuery and Bootstrap.
- Working of Web services – API based architecture with REST.
- Back-end development with Java 11.

1.1 | Introduction

As the evolution of computer science escalated rapidly by the late 20th century, computers became smaller and more powerful. However, there was no central solution to digitally connect the masses. This was not possible until the invention of the Internet in the 1970s. Earlier, the Internet was not available for public use but by the start of the 1990s, businesses began to make inroads into the world of Internet connectivity. The advent of the Internet to the scene meant that physical shops were transformed into digital entities. Today, businesses and governments use the Internet to facilitate their operations.

The Internet mainly relies on the World Wide Web (WWW), also known simply as the Web, which uses HyperText Transfer Protocol (HTTP) to distribute information across various networks. This information is entailed in websites which are formatted in HyperText Markup Language (HTML). In the last three decades, the design, structure and capabilities of websites have changed a lot. From static websites to dynamic websites, and from dynamic to responsive websites catering to the demands of mobile users, there has been a great deal of change in the web sphere.

All these modifications were made possible due to the use of several web technologies. With time, many technologies such as Java's Applets and newer ones such as Node.js came to the scene. Today, full stack web development is the contemporary practice to develop websites.

In this chapter, you will learn several concepts that will give you an idea about the full stack development.



At the time of writing this book, there are around 710 coding languages in the world.

1.2 | What is Full Stack Web Development?

In order to understand the full stack web development better, we can study a real-life example of an e-commerce application. Let us explore Amazon.com, a popular e-commerce platform known across the world for its innovative delivery mechanism. Amazon.com is widely accessible via a website www.amazon.com or via a mobile application. Both these platforms are known as *front-end* or *client-side*. These front-ends connect with other application that resides somewhere in a remote cloud server. This application is called *back-end application*. It provides services that can feed data to the front-ends. Front-ends can be developed in HTML, CSS, jQuery, Bootstrap, etc., and back-end can be developed in Java, C#, Python, Ruby on Rails, Node.js, etc. Since amazon.com is a private company and its platform is a proprietary one, it is difficult to predict the technologies they use.



1.2.1 Front-end

Usually, to design the view (or the client-side) of a web page which is visited by users through web browsers, we have the front-end development where HTML, CSS, and JavaScript are the fundamental technologies. All the menus, sliders, labels or anything you click or read on a website are generated with the help of front-end. It is all about graphics, how everything appears to users. Hence, front-end web development is also called *web design*.

1.2.2 Back-end

There is also a back-end, which is also known as the server-side. The back-end is used for business logic. It consists of a server which receives requests, an application which waits for requests and generates a response, and a database which stores all the data.

Traditionally, front-end and back-end are handled by separate professionals who have mastered any of these fields to power the website.

Full stack web development is a practice in which both the front-end and back-end are managed by the same professional. A full stack developer is not a master of a single domain; they have the conceptual knowledge and the technical expertise to create both the front-end and the back-end of websites from scratch. Such developers generally have experience in both domains, enabling them to gain an all-round understanding about all layers of web development.

The back-end contains two parts: (a) application layer and (b) persistence layer, also known as database layer. Figure 1.1 shows the three-tier architecture in which the front-end is referred as *client tier* and the back-end is divided into *application tier* and *database tier*. The application tier contains business logic and the database tier stores data generated by the application or entered by a user from the front-end tier.

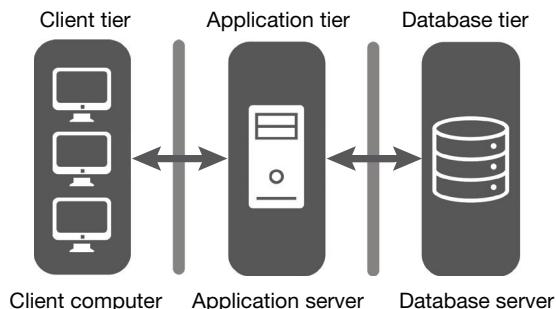


Figure 1.1 Three-tier architecture.

1.2.3 Technologies Essential for Full Stack Development

In order to become a sought-after developer, mastering the relevant technologies is essential. As we have learnt earlier, to work as a full stack developer, you need to learn technologies for all the layers shown in Figure 1.2.



Figure 1.2 Technologies for full stack application development.



1.3 | Introduction to Web Application Development

A web application exists on a server and is used when it is opened via web browsers. Web application development is the production of application programs that are run on servers and provide useful functionalities to the end-user with the help of the Internet. They include both client-side and server-side programming. They provide capabilities such as linking with databases, giving back a response to browsers, and performing a task for a user. E-commerce, social media websites, online banking are some examples of web application development. HTML, CSS, and JavaScript are some commonly used technologies for the front-end of a web application development. As shown in Figure 1.3, these technologies fall into the following categories:

1. Behavioral, which deals with behavior of an application such as actions performed on various events
2. Structural, which deals with forming the structure of an application such as adding tables, titles, etc.
3. Presentational, which focuses on the look and feel of an application like color, font style, alignment, etc.

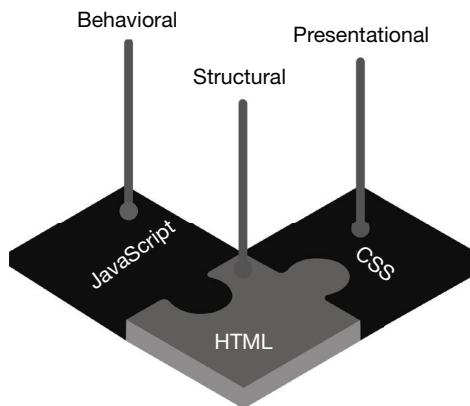


Figure 1.3 Front-end technologies for web development.



In 1999, Dale Dougherty initially coined the term “web development”. Later it was popularized by Tim O'Reilly and Dale Dougherty in late 2004.

1.4 | Front-End Technologies

A tool to run front-end comprises numerous things – it can be a computer, laptop, mobile phone, tablet, smartwatch, car dashboard screen, etc. Although the development of each of these front-end types would be different, the basic concept of design decisions remains the same. You need to understand your end-user's expectations in order to design applications. The main aim of the front-end is to provide a user-friendly interface to allow interaction with backend and database. As a full stack developer, you must understand human-computer interaction (HCI). This field of study focuses on all aspects that ensure that the system is usable. In other words, HCI focuses on improving the design of computer systems which makes it easier for users to interact with them. However, since HCI is a vast area of study, we will not cover this topic entirely in this book. Figure 1.4 helps us to understand the elements that are involved in HCI.

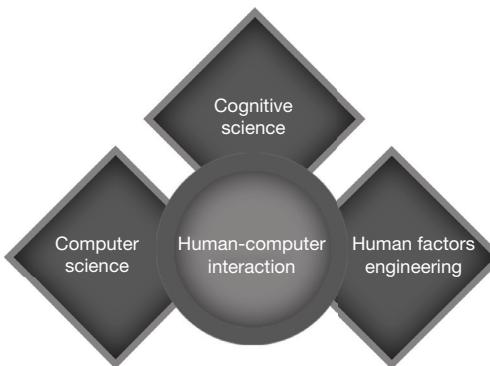


Figure 1.4 The multidisciplinary field of HCI.

In this book, we will study web-based application development. Hence, our front-end development focus will be on HTML, CSS, jQuery, and Bootstrap. Now, let us move on to understand these technologies a little bit. We will be covering each technology in detail in its own chapter, so for now, let us just introduce these briefly.

1.4.1 HTML/CSS

You must have a proficiency in HTML, which is an irreplaceable component of web development. HTML creates the arrangement of components of a web page with the help of markup. Markup powers HTML elements to use text and present it in a certain way. For instance, it can make some information on the page to appear “bold” on a website. The World Wide Web Consortium (W3C) is an organization that carries the responsibility to update it according to the ever-changing web scene.

As an example, let us see the following code which provides a title, heading, and a paragraph. While we have used Sublime Text 3 to write, one of the best things about HTML is that it does not require separate software. For learning purposes, you can even use notepad. However, in real-world development, web developers generally use an editor as it comes with lots of useful functionalities and plugins, which saves time and improve productivity.

```
<!DOCTYPE html> html
<html>
  <head>
    <title>What Is Full Stack Web Development?</title>
  </head>
  <body>
    <h1> My Heading: Full Stack Development</h1>
    <p>In this book we will learn about Full Stack Development.</p>
  </body>
</html>
```

You can use any text editor like notepad to paste the above code and save this file as “whatisfullstack.html”. In this example, we are saving this file with “.html” extension. This extension is used to save an HTML file, which can be opened in any web browser. All browsers know how to render an HTML file. This way any browser can render this code and produce a result. If you run this file in any browser, you would get the following response.



However, HTML elements are not too visually appealing. This is where Cascading Style Sheets (CSS) come into the picture. As the name suggests, CSS adds “styles” to HTML components. Therefore, CSS is heavily used to provide striking visuals to the web pages for incorporating layouts, designs, and variations.

While HTML did have its own features to add style to web pages, it became too complex and unmanageable to add colors or fonts to each web page separately. Hence, CSS gained recognition, where all the styling-related work was administered through separate style sheets. As an example, let us add red color to our previous HTML example and also put it across the center.

```
<!DOCTYPE html>
<html>
  <style>
    h1 {
      color: red;
      text-align: center;
    }
  </style>
  <head>
    <title>What Is Full Stack Web Development?</title>
  </head>
  <body>
    <h1> My Heading : Full Stack Development</h1>
    <p>In this book we will learn about Full Stack Development.</p>
  </body>
</html>
```

You can either add this code in the same file or create a new one. For keeping these two examples separate, we have created two different files. You can just add style code above `<head>` tag and save.

```
<style>
  h1 {
    color: red;
    text-align: center;
  }
</style>
```

Upon opening this file in a browser, you will see the following image



Sir Tim Berners-Lee, a physicist at CERN and the inventor of the Internet, first proposed and prototyped ENQUIRE, a system for CERN researchers to share documents. Later in the year of 1989, he proposed an Internet-based hypertext system and in the subsequent year (1990), defined HTML and developed browser and server.

QUICK CHALLENGE

Add more elements like labels, buttons, radio buttons, etc. and render it in a browser. Make sure the header color is blue and the text is aligned right.

1.4.2 JavaScript

After HTML and CSS, we have JavaScript – the most in-demand language of recent years in the technology community. Unlike the former two front-end technologies, JavaScript is a proper high-level programming language, a scripting one. JavaScript enables developers to add more sophisticated functionality on the client-side. It transforms a web page from a static to a dynamic one by adding interactivity and timely features such as animations, advanced maps, etc. All the common web browsers have a dedicated engine to process the language on the client-side.

Today, JavaScript has popular front-end frameworks like Angular JS, Vue JS, React JS, etc. While it was originally used for the client-side, it has come on the back-end with Node.js. For a simple JavaScript example, let us see the following. We will use the same file to add a simple JavaScript code to multiply two elements.

```
<!DOCTYPE html>
<html>
  <head>
    <title>What Is Full-Stack Web Development?</title>
    <script>
      var pagesperday = 5;
      var readingdays = 30;
      var daysneeded = pagesperday * readingdays;
      document.getElementById("result").innerHTML = daysneeded;
    </script>
    <style>
      h1 {
        color: red;
        text-align: center;
      }
    </style>
  </head>
  <body>
    <h1> My Heading : Full Stack Development</h1>
    <p>In this book we will learn about Full Stack Development.</p>
    <p>
      If you read 5 pages per day for 30 days, you will complete <span
      id="result"></span> pages.
    </p>
  </body>
</html>
```

This code will give us the following result. In this, we are doing a simple multiplication of two variables and placing the result in an HTML file which has id as “result”.



**QUICK
CHALLENGE**

Write codes to perform various other functions like addition, subtractions, etc.

For this book, we will focus on jQuery as a JavaScript library.

1.4.3 jQuery

jQuery is a small but powerful JavaScript library. It is used to ease scripting on the client-side with HTML. It can be used for traversing HTML documents or modifying them. Likewise, it is used for event handling and producing animations. The primary objective of jQuery is to add interactivity on a website. Since it can help to write shorter codes than JavaScript, therefore it is also called “write less do more”. jQuery is a cross-platform. For a simple jQuery example, let us see the following. This example adds dynamicity to HTML as it hides the heading by a simple button click.

```
<!DOCTYPE html>
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
        <script>
            $(document).ready(function() {
                $("button").click(function() {
                    $("h2").hide();
                });
            });
        </script>
    </head>
    <body>
        <h2>This element will get removed once the following button is clicked.</h2>
        <p>JQuery Example Code</p>
        <button>Hide Heading</button>
    </body>
</html>
```

Save this code in the same way you saved the HTML code earlier and run in a browser window. Upon running this, you will see the following output.



**QUICK
CHALLENGE**

Take the above example and add code to replace text from the `<p>` tag. For example, change “JQuery Example Code” to “jQuery Content Replacement Example” on the button click.

Also, in order to run this application on mobile and tablet browsers, we need to use a library known as Bootstrap.

1.4.4 Bootstrap

As Nokia was dethroned by the likes of Apple and Samsung, several smartphones and tablets were introduced in the market in the late 2000s. At that time, websites were primarily designed for desktops and laptops. As more and more people used smart devices to surf the Internet, they found the web experience to be extremely unsatisfactory. To solve this issue, Twitter's Jacob Thornton and Mark Otto developed Bootstrap in 2011; hence, you might have often read it as “Twitter Bootstrap”. Bootstrap was not a new front-end markup language, but it used the existing front-end technologies HTML, CSS, and JavaScript and paved the way for responsiveness web design.

Bootstrap is the most famous HTML, CSS, and JS framework for designing responsive, mobile-oriented projects on the web. In other words, Bootstrap is a huge collection of useful, reusable bits of code written in CSS, HTML, and JavaScript. It has become an important tool for front-end developers. Developers and designers can quickly create fully responsive websites with the help of Bootstrap.

Responsiveness is a metric which refers to the tailored rendering of a website with respect to its device and screen size. Hence, the advent of Bootstrap made it easier for web designers to make websites for a wider audience. However, responsiveness is not the only advantage brought to the table by Bootstrap. It also provides developers with templates for HTML and CSS to add tables, image carousels, modals, buttons, forms, typography, etc. on websites.

1.4.4.1 What are the Advantages of Using Bootstrap?

Using Bootstrap saves us from writing tons of CSS codes. Moreover, the best thing about Bootstrap is that it is free. Presently, Bootstrap is hosted on GitHub. Following are some of the other advantages of Bootstrap:

- 1. Responsive grid:** There's no need to spend hours to code your own grid. Bootstrap provides you with its own grid system. By using it, developers can directly fill their containers with content. Setting up your custom breakpoints for each column is super easy by using their small, medium, and large breaks. You can just choose the default as it could already meet your requirements.
- 2. Responsive images:** Bootstrap helps you to automatically resize and optimize according to the screen size by using its own code. Furthermore, you can also change the shape of the images. This could be easily done without continuously switching between the code and your design software.
- 3. Components:** Bootstrap provides a handful of components which could be tracked onto your web page. This includes:
 - Navigation bars.
 - Dropdowns.
 - Progress bars.
 - Thumbnails.
 Not only will you be able to add eye-striking elements to your web page, but you will also be able to optimize them automatically according to different screen sizes. You do not need to do tons of work as there are a lot of ready-made functionalities are ready to use.
- 4. JavaScript in Bootstrap:** Bootstraps provides developers with tons of jQuery plugins. jQuery facilitates with a greater level of interactivity. This creates easy solutions for modal popups, image carousels, transitions, etc.
- 5. Documentation:** The document of Bootstrap is one of the best documentations in the software industry. Each piece of code is well described to the smallest detail in their official websites. These explanations also include code samples making it perfect for the beginners. You just need to copy a code and paste it into your page.
- 6. Bootstrap community:** Just like any other open-source software, Bootstrap also has a very large and friendly community of developers and designers. Developers find it easy to modify and contribute to the Bootstrap's database as it is hosted on GitHub. People often collaborate, give useful pieces of advice, and interact with each other to solve doubts.

For a simple example, let us create the following table in Bootstrap.

```

<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Bootstrap Example</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
    </head>
    <body>
        <div class="container">
            <h2>Table Example</h2>
            <p></p>
            <table class="table">
                <thead>
                    <tr>
                        <th>Student Name</th>
                        <th>Course Marks</th>
                        <th>Total Marks</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>Merry</td>
                        <td>Introduction to Computer Vision</td>
                        <td>95</td>
                    </tr>
                    <tr>
                        <td>Jim</td>
                        <td>Distributed Computing</td>
                        <td>63</td>
                    </tr>
                    <tr>
                        <td>Morrison</td>
                        <td>System and Software Security</td>
                        <td>74</td>
                    </tr>
                </tbody>
            </table>
        </div>
    </body>
</html>

```

This program will produce the following result upon running in a browser window.

Student Name	Course Marks	Total Marks
Merry	Introduction to Computer Vision	95
Jim	Distributed Computing	63
Morrison	System and Software Security	74

Using a simulator add-on for browsers, we can see how this code renders on different devices. Following are views on some devices.

Student Name	Course Marks	Total Marks
Merry	Introduction to Computer Vision	95
Jim	Distributed Computing	63
Morrison	System and Software Security	74

iPhone X View

Student Name	Course Marks	Total Marks
Merry	Introduction to Computer Vision	95
Jim	Distributed Computing	63
Morrison	System and Software Security	74

Samsung Galaxy S5 View

Student Name	Course Marks	Total Marks
Merry	Introduction to Computer Vision	95
Jim	Distributed Computing	63
Morrison	System and Software Security	74

iPad Pro View

**QUICK
CHALLENGE**

Create a new HTML file and write code to create four div elements. Use CSS to show them in four columns. Add an image element in each div. Use Bootstrap to make all four divs render properly on devices like iPhone, Samsung Galaxy, and iPad.

1.5 | Back-End Technologies (Server-Side)

The server-side requires the use of a single or multiple programming languages to write business logic, user authentication, or database (DB) related work. This is a matter of preference. Each back-end language has its pros and cons. There is no back-end language that is the best solution for all needs; each back-end language is good to solve a certain problem. As a computer programmer, you must be able to adapt because computer science always meant to be for “solving problems” rather than “solving problems with a specific language”. We have the following options as back-end languages:

- 1. PHP:** Easily the most common tool for designing the back-ends of a website. PHP is good for those who want convenience at the expense of functionality. Setting up websites with PHP requires minimal effort and time. It is generally used with frameworks such as Laravel, CodeIgniter, etc. However, there are various content management platforms such as WordPress, Joomla, etc. that are popular to run websites. These frameworks are customizable with plugins.
- 2. Node JS:** This allows developers to write both the front-end and back-end in a single language – JavaScript. This is a great option for those who are proficient in the foundations of JavaScript, though there are still questions about Node JS maturity and capability to work in enterprise-level applications. Node JS is often used in a technology stack known as MEAN (short for MongoDB, Express, Angular, Node JS).
- 3. Python:** This is increasingly becoming an all-purpose language. It is used heavily in Artificial Intelligence and Data Science for making computers “intelligent”. Some also use it for desktop development as well as design management systems. Similarly, the web sphere has also accepted it. Python’s charm comes with its ability to allow users to write only a few lines of code. While a developer may have to write a simple “Hello World” program in other languages in multiple lines, Python can execute it in a single line. Python is generally used in Web with Django and Flask frameworks.
- 4. C#:** This is the brainchild of Microsoft, one of the giants ruling over the IT industry. C#, along with Java, is one of the two languages that is primarily used for coding enterprise-level development. C# itself is inspired from Java. Due to Microsoft’s R&D, the language is used heavily across different businesses and industries where it is the core programming medium of “Microsoft” stack, which runs Microsoft OS like Windows Server, databases like SQL Server, and C# to design desktop application with Visual Studio or writing web applications with the ASP.NET framework.
- 5. Java:** Java is perhaps the most used language in the world. It is used all around the world for banking ecosystems because of its high security. Universities use it to teach CS courses such as Object-Oriented Programming, Data Structures, etc. The world’s leading mobile operating system Android depends on mobile apps which are written in Java. On the web, Java uses a multitude of frameworks such as Java EE, Play Framework, Vaadin, Struts, etc. However, the most famous one is Spring Model View Controller (MVC).



Java programming language was originally named Oak. However, Oak was already taken by another company, which made the Java team to come up with a new name.

For this book, we will select Java as the back-end development language. Java has many advantages over other programming languages and has received huge support from the business world. It is the by default choice for many large-scale applications and has improved a lot over the years to keep its leading position. Let us study the latest version of Java, most of which we will learn in detail in later chapters.

1.6 | Introduction to Back-end Development with Java 11

There is a reason Java has cemented such a strong position in the global community over the past few decades. Its cross-platform computability with Java virtual machine (JVM) was a breakthrough that gave birth to the likes of Scala today. Some credit its success to its massive community while many believe that Java has a library for anything. Experienced programmers opine that it is actually the scalability and security factors that have made the language invaluable. Let us take for example Twitter, which

was designed using Ruby on Rails but broke at a point and had to adopt JVM for avoiding the crisis again. There are similar examples available throughout the industry where Java proved to be a lifesaver in adversity.

However, Java is not without its detractors. Earlier Java EE proved to be messy and complex. It was not the apparent vulnerability of the language that proved to be the negative point, but it was actually its difficulty level to work with that discouraged others. With Java EE (then J2EE), the first issue was working with a component which needed the configuration of XML files. The second issue was setting up the component dependency (e.g. whenever a component, say X, had to use a different component, say Y, then component X had to search by itself for component Y). Lastly, some services were not required by the components. Hence, there was the problem of “heavy weight”.

Fortunately, with the release of modern-day Java frameworks such as Spring, Spring Boot, Play, etc., these issues have been addressed well.

Today, Spring is considered by many to be the leading Java framework. It is a lightweight framework which can be used for a plethora of web projects. Spring is known for working with a concept called *dependency injection*. Similarly, aspect-oriented programming is another fundamental concept of the framework. These concepts have been covered in detail in Chapter 20.

Spring has proved to be effective in removing boilerplate code and has been praised for its reduction of complexities, which proved to be a nightmare for developers working with Java EE.

Spring can be seen as a framework which provides integration of all Java API and technologies and enables their usage with plain old Java objects (POJOs). Hence, it is important to realize that Spring does not completely invent new technologies in Java's web framework. Instead, it gives a powerful and optimized solution through which technologies such as EJB, JMS, and Hibernate can be used easily.

Due to its wide support to integrate a long list of technologies, Spring framework is used for the creation of versatile, testable, and efficient code which is good for both traditional ecosystems and newer ones with Android or functional paradigms. Now, let us explore the MVC pattern and see how Spring MVC will be useful for the development.

1.7 | Introduction to Model View Controller (MVC)

Model View Controller (MVC) is an architectural pattern that is used to build software applications. Due to its effectiveness, the pattern has received huge adoption in the web space where you can see Spring MVC and ASP.NET MVC as one of the most well-known proponents of it. The pattern works by dividing an application into three layers, which help improve modularity and reusability of the application while they are more flexible and can support iterations.

To explain MVC, let us consider a simple example. You go to a coffee shop for your morning breakfast. You interact with a barista to order an Espresso. The barista communicates your order to others where a manager makes sure that your order is initiated and processed properly. The coffee machine and the staff use your information and make the right coffee. Afterward, the barista gives you the coffee. Here, the manager who masterminded everything is the Controller. The coffee machine and the staff who had the information can be referred to as the Model, while the barista who was on the “front” can be called as the View. Now going by this let us explain what each layer does.

- Model:** It carries out all the management of all the data-related logic. Likewise, the DB operations like SELECT, UPDATE, are performed at this layer. Mostly, it engages in communication with the controller while at times it may directly provide some data to the view.
- View:** It is the user interface (UI) which is seen by the users. All the front-end codes like HTML, CSS, and JavaScript are part of the view. The controller can forward dynamic values to the view for real-time updates.
- Controller:** It gets user-input from either a URL request or a view. It then goes on to perform actions on the requests. The view receives data from the controller while the model provides that data to the controller.



What is the importance of Model in the MVC framework?

1.8 | Introduction to Web Services: API-Based Architecture with REST

As each programming language possesses its own strong and weak points, a business may use PHP to build one web application while Java may prove to be more practical in another scenario. At times, there is a need to establish communication between these web applications that exist on separate technology stacks. This is where web services come into play. A web service is an



intermediary that is used for exchanging messages between the server and the client operating on the Web. Web services are programmed to execute multiple actions.

1.8.1 REST

REST is a type of web service; it competes with another type of web service known as Simple Object Access Protocol (SOAP). Some analysts find SOAP to be more powerful while some credit REST as it needs a lesser amount of bandwidth, thereby making it extremely suitable for the Internet.

Application Programming Interface (API) refers to lines of code that facilitate two different programs to engage in communication. API carves out a methodology for the coder through whom a program may go on to ask for services from any application or OS.

RESTful APIs work by decomposing a transaction into several little modules. Each of these modules is made to solve a different portion of the transaction. This may be a bit complex for developers to code but it reaps rewards with an enhanced degree of modularity. RESTful APIs use HTTP protocol and its standards. Such an API may use GET for the retrieval of a resource. Similarly, it can use PUT for making an update or change in a resource. Likewise, it may generate a resource by POST or eliminate it by utilizing DELETE.

In REST architecture, calls do not carry any state. Thus, they are a good fit for cloud-based environments. Components that are stateless are advantageous because they can be easily redeployed at any failure while they can also support scaling for load changes. The reason for this is that requests can be easily sent to any of the component's instance.



Why is modularity important in developing a scalable application?

1.8.2 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a messaging protocol that makes it possible for the distributed elements of an application to communicate with each other. SOAP uses eXtensible Markup Language (XML) to enable communication among web services. Its main aim is to facilitate neutrality, extensibility, and independence. It can also be used for broadcasting a message.

SOAP facilitates processes operating on disparate operating systems (e.g. Windows and Linux) to interact using XML.

1.8.2.1 Characteristics of SOAP

Following are the main characteristics of SOAP:

- 1. Neutrality:** SOAP can function over any protocol such as HTTP, SMTP, TCP, UDP, and JMS.
- 2. Extensibility:** Security and WS-Addressing are one of the top priorities of SOAP.
- 3. Independence:** SOAP allows any kind of programming model.

Even though SOAP can be utilized in a plethora of messaging systems and can be provided by a plethora of transport protocols, the main focus of SOAP is remote process calls transported through HTTP. Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein designed SOAP as an object-access protocol for Microsoft in 1998. However, this new specification was not available publicly and it was submitted to the Internet Engineering Task Force (IETF) for review on 13 September 1999.

SOAP has a very rigid structure and a message must be formatted in the required format which contains Envelope, Header, Body, and an optional element Fault. SOAP message is an ordinary XML document which contains the following elements:

- 1. Envelope:** SOAP envelope is a mandatory element. It is just like a regular mail envelope that holds the letters we try to deliver.
- 2. Header:** This element is optional. It is used to send attributes that can be useful for processing the message either at the intermediate point or at the end-point.
- 3. Body:** This is another mandatory element which contains the actual data that is being sent in XML format.
- 4. Fault:** This is an optional element which is used to provide information about message processing errors.

The following code shows the general structure of SOAP and contains the above mentioned elements:

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2003/05/soap-envelope"
    SOAP-ENV:encodingStyle = "http://www.w3.org/2003/05/soap-encoding">
    <SOAP-ENV:Header>
        ...
        ...
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        ...
        ...
        <SOAP-ENV:Fault>
            ...
            ...
        </SOAP-ENV:Fault>
        ...
    </SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

1.9 | Communication Between Front-End and Back-End

Front-end and back-end must use a mechanism to communicate with each other. In order to get data from web services, we need to send a request from front-end and receive a response from the back-end. In the following subsections we will highlight two formats that we can use to send and receive information between front-end and back-end.

1.9.1 JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a form of syntax which is used to store and transfer data on the Internet. It is plain text which takes advantage of the object notation from JavaScript. A browser and server can only send and receive data which is in the textual form. With JSON, we have the data in text. Objects in JavaScript can be easily transformed into JSON, which are then used by the server. The server can then convert JSON into an object of JavaScript. This means that there is no need for parsing.

In more formal terms, it can be said that JSON is a representation of data which does not operate on the back of schema and uses ordered lists and key-value pairs to manage data. Due to its convenience, JSON receives support from all common programming languages. Today, it exists as one of the primary mediums for the exchange of data by clients and servers in the web industry. As a basic example, let us take a look at the following program:

```
var adam =
{
    "occupation": "software developer",
    "city" : "Austin, TX",
    "country": "USA"
};
```

This program generates an object which can be accessed by “adam”. The curly brackets represents the “value” of our object. An object can entail multiple properties by the use of key-value pair, where commas demarcate them. In order to get the value of any property of “adam”, we can write the following:

```
document.write('Adam lives in' adam.city);// In return the output is Austin, TX.
document.write('The country of residence for Adam is' adam.country);// In return the
output is USA.
```

A single variable can also store information for multiple people. To do this, square brackets are to entail more than one object. For example, for two employees, see the following program:

```

var employees = [
  {
    "name": "Adam",
    "city" : "Austin, TX",
    "country": "USA"
  },
  {
    "name": "William",
    "city" : "Fort Worth, TX",
    "country": "USA"
  }
]

```

1.9.2 Extensible Markup Language (XML)

Extensible Markup Language (XML) is commonly used to describe data. Its format is flexible which assists to generate various formats for information while it is utilized to exchange data on enterprise networks and public networks. XML bears some similarity to HTML as both make the use of markup. However, while HTML is used for displaying the contents of a web page, XML is used for displaying the data. The terms self-defining and self-describing are sometimes associated with XML because the data's structure is embedded in the data and therefore there are no requirements for any pre-building of the structure needed for the storage of data.

The most fundamental component of a document written in XML is known as *element*. To define elements, tags are used. All elements have an opening tag and a closing tag. The root element is the outermost element, which encompasses all the elements of an XML document. Hence, hierarchy is supported in XML. The following is a basic XML example,

```

<?xml version="1.0" standalone="yes"?>
<conversation>
  <greeting>Hello World!</greeting>
  <response>Hello Friend!</response>
</conversation>

```

1.9.3 Database

One must have a fundamental knowledge of databases. First, there are traditional relational databases where tables are known as *relations*. Each relation has a row (or record) and a column (or attribute) to store the details about the tables. MySQL, SQL Server, and Oracle are some of the most in-demand databases for storing data. These databases use a domain-specific language known as Structured Query Language (SQL) for writing statements.

Second, we have the latest NoSQL databases. NoSQL databases are unique because they do not use SQL; in fact, NoSQL stands for “NotSQL”. NoSQL databases provide a greater degree of flexibility and operational speed, but they compromise on data consistency.

In this book, we will learn to use MySQL database. Also, in order to connect to this database, we will use an Object Relational Mapping (ORM) tool known as Hibernate. Let us first understand ORM with Hibernate.

1.10 | Introduction to Object Relational Mapping (ORM) with Hibernate

The connection between relational models and object models often results in complexities because of their unique approaches. While object-oriented programming (OOP) languages such as Java use an interconnected object graph to represent data, relational database management systems work with data in a tabular format. To solve this issue, ORM came into existence. ORM allows developers to access and modify objects while it saves them from thinking about the relation of objects with data sources.

By using the OOP concept of abstraction, ORM maps details of RDBMS or XML data sources with a single or multiple objects where the updated modifications in the linked interfaces are kept hidden from the programmers.

This means that all the encapsulation happens inside of ORM. Hence, ORM updates itself when there is a change in the API or the data sources instead of letting the application do the hard work. Therefore, developers find it convenient to use newly available classes. Figure 1.5 shows how objects in memory are linked to RDBMS via ORM's mapping logic; a simple representation to understand the ORM concept.

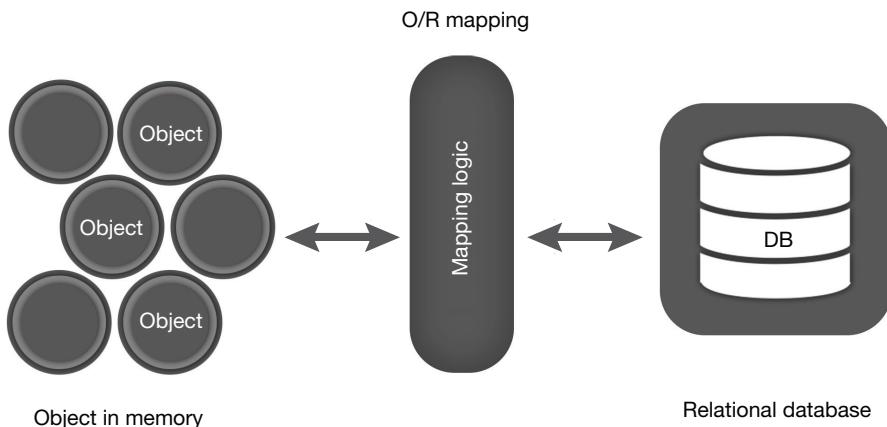


Figure 1.5 ORM mapping.

1.10.1 Hibernate

Hibernate is one of the most popular tools for ORM in Java's ecosystem. It facilitates applications to gain persistence. Persistence refers to the phenomenon in which the data in applications lasts longer than the processes of the application. This is useful when programmers desire to extend the life of any object from out of JVM's scope so they can reuse it later. As an ORM tool, its converts POJOs to tables of relational databases.

Summary

As you must have understood, in order to develop a meaningful application, one should have knowledge of the complete development stack. Since the technology is continuously evolving at a faster rate, there is a huge demand for full stack developers who cannot only understand the overall development challenges but also help in identifying the right technologies for the required purpose.

The aim of this book is to give you a practical path to learn full stack development. This will give you hands-on experience while learning the concepts. Hence, we will be taking you through a journey of developing a real-life application to see how the concepts can be applied in practice.

In this chapter, we have learned the following:

1. Meaning of full stack development.
2. Essential skills and technologies required by a full stack developer.
3. Web application development and various front-end technologies such as HTML, CSS, JQuery, and Bootstrap.
4. Back-end technologies and Java 11.
5. MVC, JSON, and XML and how to use them.
6. Web services and end-point APIs.
7. How to think of creating end-points.
8. Databases and how to use of them in the application development.
9. ORM and concept of Hibernate.

In the next chapter, we will discuss a project idea and plan out its development path. We will learn about various architectural styles and tools.

Multiple-Choice Questions

1. Which one of the following objects returns the present and updated position of the user when the user tries to make a move from one position to other?
 - (a) `getCurrentPosition ()`
 - (b) `clearWatch ()`
 - (c) `Timestamp`
 - (d) `WatchPosition()`
 2. Which one of the following methods will help you capture all the click events in a window?
 - (a) It won't be possible to capture the events
 - (b) `window.routeEvents () :`
 - (c) `window.raiseEvents (Event.CLICK);`
 - (d) `window.captureEvents (Event.CLICK);`
 3. _____ is also called the contaminated property of a window object in JavaScript.
- (a) Protocol
 - (b) Pathname
 - (c) Host
 - (d) Default Status
4. A unit of a JavaScript must start with and end with _____.
 - (a) Semicolon, ampersand
 - (b) Semicolon, colon
 - (c) Ampersand, colon
 - (d) Ampersand, semicolon
 5. It is possible to embed JavaScript code with HTML directly.
 - (a) True
 - (b) False

Review Questions

1. What is full stack development?
2. What are the uses of HTML and CSS?
3. How should one add Table to an HTML page and make it responsive to render properly on major devices?
4. What is the use of JavaScript?
5. Why should we use jQuery instead of plain JavaScript?
6. What is the benefit of MVC pattern?
7. What are the important elements of web development?
8. What is ORM?
9. What are the benefits of using Hibernate?
10. Why should we use web services?
11. What is REST web service?
12. What are the advantages of REST versus SOAP?
13. Why is Java better than PHP for back-end development?
14. Give five reasons why developers should use Bootstrap.

Exercises

1. Create an HTML page that shows a student report. Use tables, headings, etc.
2. Add CSS to make it look good and use different colors, font sizes, etc.
3. Add Bootstrap to make sure it renders properly on all major devices.
4. Add jQuery code to add a new record to the table.
5. Create a detailed comparison chart for JSON and XML.

Project Idea

Problem Statement: A school in a village has about 400 students in various classes. They do have Internet and electricity connections. However, the problem is their students come from far distances. Hence, they want to develop a web application which will allow them to add e-learning courses and exercises so that students do not have to come for extra classes. Design an e-Learning portal where the school can add courses and students can log in and access these courses.

Draw three-layer architecture diagram like front-end, back-end and database for the above-mentioned problem. Think of HTML pages you will need to complete the tasks and decide on database tables you will need. Also, decide on a communication technology (such as JSON or XML) you will use and explain reasons behind your selection.

Recommended Readings

1. Jennifer Preece, Helen Sharp, and Yvonne Rogers. 2015. *Interaction Design: Beyond Human-Computer Interaction, Fourth Edition*. Wiley: New Jersey
2. Mex Tegmark. 2008. *Life 3.0: Being Human in the Age of Artificial Intelligence*. Knopf: New York
3. Paul R. Daugherty. 2018. *Human + Machine: Reimagining Work in the Age of AI*. Harvard Business Review Press: Massachusetts
4. Nick Bostrom. 2016. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press: Oxford

Getting Started with Full Stack Development: A Project Idea

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- E-commerce and its advantages.
- Entity Relationship Diagram (ERD) and an entity.
- How to create a flowchart from the development perspective.
- Planning and execution of a development project.
- The designing of front-end structure and interfaces.
- The designing of web services APIs and the definition of required endpoints.
- GET and POST methods to send data from client to server.

2.1 | Introduction

This book is intended to be a practical guide to full stack development. It not only covers the theoretical part but also focuses on the practical use. Keeping this objective in mind, we will be learning through an example application. As we learn about the technologies, we will apply this newly gained knowledge to design a real-life application. In this chapter, we will introduce a project idea and start building it. For this exercise, we will use an e-commerce example. For those who are new to the concept of e-commerce, we will explore this in detail as well.

2.2 | Project Outline

This project has been specially designed to take you through the full development process of front-end web development and back-end scalable web service endpoints. We will be considering all the major functionalities of an e-commerce website such as a shopping cart, customer relationship management (CRM) integration, payment gateway, etc. We will think of all the design choices from the user's point of view and consider security and performance.

Before we begin with the project planning and technical diagrams, let us first understand the e-commerce domain.

2.3 | What is E-Commerce?



E-commerce is the process of buying and selling of goods through electronic means (Internet or apps). Over the last decade, the popularity of e-commerce has hugely increased and it is replacing traditional brick and mortar stores. E-commerce helps us buy products on a global scale anytime during the day or night.

2.3.1 Advantages of E-Commerce

Following are the main advantages of e-commerce:

1. **Low start-up cost:** Physical stores have to spend tons of money to create an establishment and buy inventory, sales equipment, and more. Plus, having a physical store means that you have to pay your staff to work. On the other hand, starting up an e-commerce is quite cost-effective. You just need to spend on the website or app development, a tie-up with a postal service, and inventory. This naturally helps the entrepreneurs to keep the costs down.

2. **24/7 chances of income:** When you open a traditional store, you have to decide particular opening and closing hours. You cannot simply keep your store open all the time as it will need you to spend more resources on your store. Having an e-commerce platform means you are always ready for business. The web is already there for customers to navigate to your website. Thus, you do not need to have employees to be working in the night to process the orders.
3. **Sell anywhere:** Compared to traditional stores, you can sell products anywhere depending on the reach of the courier service. However, having a shop means you will have only one location until and unless you open other branches (which is expensive).
4. **Get access to customer data:** Building a relationship with the customer is very important to boost sales. In traditional stores, the customers are often uncomfortable to give away their personal details like phone number, address, e-mail, etc. Online e-commerce makes this simple by creating user accounts for the customers where they need to put their personal details to process orders. You can even conduct marketing surveys for products in your inventory with the help of e-commerce.
5. **Helps process a large number of orders at once:** Surely you can manage a large number of orders in a traditional shop. However, you will need a large workforce and space to accommodate such power. E-commerce makes it so simple and automated that you really do not need to do much. In e-commerce, the customers can easily place single or multiple orders according to their own schedule.
6. **Easier to encourage buying:** What looks the best sells the best. Obviously, you can decorate your traditional store with decorations and place your products smartly. However, you will be spending a lot on just creating a good presentation. On the other hand, in e-commerce, everything is much easier by creating a vivid, fluid, and intuitive website design.

2.3.2 Important Considerations while Developing an E-Commerce Website

There are quite a few factors to take into account while developing an e-commerce website. Some of the important ones are discussed below.

1. **Knowing target customer base:** Before we begin designing an e-commerce website, it is important to understand the target market. This will tell us how large a market we are serving so that we can design our application accordingly. This information helps in selecting the technologies we will need. For example, for a larger customer base like Amazon.com, our traditional RDBMS database will not give good results as Amazon sells millions of products in a short amount of time. For this, we need to consider NoSQL databases like Cassandra which are faster than RDBMS databases. Also, knowing our customer base will tell us what other features we need to consider. For example, for a younger target market, we need to accommodate social media marketing module where the backend system will generate social media posts, track user actions, build personalized liked categories products, etc. This type of analytics is needed to make the web application more user-focused. We will also need to consider adding algorithms to help users in buying and product selections by presenting relevant products, offers, services, etc.
2. **Choosing the right technologies:** Once we know our target market, it is important to select the best possible technology stack for the development. We have already discussed the RDBMS and NoSQL databases and their comparison; there are other areas which are important in terms of technology selection. For example, product selection, offers, and promotions will be different for mobile users than web users. Hence, in a younger target market case, we need to select technologies which support mobile front-end like push notifications, real-time tracking, location-based product selection, etc. We also need to consider web servers we will be using for production deployment. Every server out there has pros and cons, so we need to weigh the selection criteria based on our target market use. The strength and weaknesses of each platform are quite different from each other. Also, our development framework might change according to server selection like configuration files, libraries, etc. Hence, a detailed analysis of these web server platforms is necessary prior to development phase. This will lower the scope for mistakes and errors. Here are a few things we should consider while selecting a technology stack:
 - CRM, ERP, database integration and compatibility.
 - Scope for e-commerce website customization.
 - Payment gateway integration.
 - Robust back-end support.
 - Mobile friendly and responsive e-commerce.
 - E-commerce marketing.

3. Design elements of the application: It is important to understand that there are various other elements that are equally important as look and feel. While designing an e-commerce application, it is extremely vital to consider performance, security, accessibility, scalability, etc. of the platform. Here are a few other things we should consider while designing the application:

- **Speed:** The users do not like to wait for long while websites get load. Search engines like Google further lower down this. Thus, speed is power.
- **Uptime:** We must focus on the uptime records of our application. Having a great uptime is essential for an e-commerce application as it is the main purpose of having an online shop, so that customers can shop any time from any place. This is an important step while considering a web server to make sure it is compatible with our technology stack.
- **Support:** We must consider customer support module while designing the application. People will not use the platform if they are not able to communicate with the support associates for any queries or help. Many users may also need support in product selection, buying choices, etc.
- **Scalability:** E-commerce applications can see seasonal surges in traffic and hence it is important to make sure that the servers are capable of handling a large amount of traffic. We must consider this while writing code as many performance problems occur due to bad code.
- **Security:** This is another vital area to consider as people will not like to get their credit card information in the wrong hands or leak their personal information. Having a secure application is crucial to building trust among the customers. There are various steps we could take while writing code, configuring servers, communicating with third-party applications like CRM system, Payment Gateways, etc. So focusing on security is essential.

4. Thorough testing: Testing should be an essential part of the development. We should not wait until the completion of the development; rather we should consider testing our code in units. This is where unit testing comes handy. We must consider writing unit tests for our code so that we find problems as early as possible in the development. Unit tests can also expose any security or performance leak in the application. Once the application is complete with successful unit tests, we still cannot directly make it public without taking it through a quality check process. Intense testing should be done in order to check usability, customer convenience, bugs, and ultimately a good shopping experience. The testing can be carried out in various web browsers, platforms, and multiple devices. This testing could be manual or automated. It can even be the combination of the two forms. We must examine for slow loading of webpages, poor navigation, broken links, and bugs to provide a seamless experience to the user. Our aim should be to catch the bugs and flaws as early as possible in the development phase. If these bugs pass the development phase they would become very expensive to manage. So what exactly should be tested during the testing phase?

- **User flow of the website:** We must check each and every page accessible to the users. Starting from the home page to the menu, product listing, search bar, and results. Give the best attention to the payment and checkout window.
- **Functionalities of the website:** This includes the categories, the pages, user registrations, filters, shopping cart, payment options, shipping options, and all other important features of the website.
- **Security of the website:** Security is vital in an e-commerce website. This applies not just to the payment page where the customers enter their sensitive information such as credit card details, but also to every page of the website. Having a secure website builds trusts among users.
- **Testing the compatibility of the website:** Everyone has different devices and different browsers while browsing the web. Thus, we must test the website on different web browsers, different platforms, and multiple devices such as mobile/tablets/laptops.
- **Performance and discoverability of the website:** No one waits for ages to open a website. Thus, it is very important to monitor the loading time of your website. Try to catch the broken links, check the SEO ranking in different search engines.

2.4 | Required Entities

Following are some of the essential entities we will need to consider in developing the end-to-end e-commerce application. We also need to identify the relationships between these entities to make sure we consider them properly in the design.

1. **Customer:** This is the main entity in the e-commerce application. Customer is the one who buys products from the e-commerce shopping portal.
2. **Address:** This entity is related to the customer entity to store multiple addresses for a customer.

3. **Currency:** This entity is to define the preferred currency for the customer.
4. **Product:** This entity represents physical or digital goods to be sold on this e-commerce platform.
5. **Shopping cart:** This entity defines a place where the customer stores his/her chosen products.
6. **Coupon:** This entity defines discounts on products.
7. **Offer:** This entity specifies promotional offers for products.
8. **Vendor:** This entity sells products on the e-commerce platform.
9. **Order:** This entity defines a consolidated place for chosen products to be purchased by a customer.
10. **Shipping:** This entity defines the product shipment process for customers.
11. **Payment:** This entity is for financial transactions that took place in a purchase process.
12. **Invoice:** This entity is a bill generated for an order.
13. **Inventory:** This entity defines a place to store product quantity-related information.
14. **Catalog:** This entity defines a ledger that shows products, offers, and discounts related details.
15. **Warehouse:** This entity provides a physical storage place for products.

QUICK CHALLENGE

Extend the functionality of this e-commerce platform and add more entities. Think of various different possibilities to enhance the user experience.

**2.5 | Entity Relationship Diagram**

Now let us consider these entities and figure out relationships between them. In Figure 2.1, we will try to link the entities to understand the linkage between them and learn how to handle them.

QUICK CHALLENGE

Create new entity relationship diagram (ERD) for the additional entities you have figured out in the earlier exercise.

**2.6 | UML Class Diagram**

Based on the entity relationship diagram (ERD), we can create a class diagram that will show the table relationships as per entities. Figure 2.2 also contains fields of the classes and their types.

The following explains how to read the relationships on the entities:

1. Single dash on both the entities shows one-to-one relationship between those entities.
2. Single dash on one and three dashes on the other entity show one-to-many relationship.
3. Three dashes on one and single dash on the other entity show many-to-one relationship.
4. Three dashes on one and three dashes on the other entity show many-to-many relationship.

QUICK CHALLENGE

Extend this class diagram with the new entities you have identified in the earlier exercise.

**2.7 | Flowchart**

Flowchart is an important element in designing an application because it makes us think of all the possibilities in the user flow. It shows each and every step of the application and how data and decisions are flowing within modules. Before we take a look at various flowcharts, let us first understand the elements and their meanings. Following are a few elements of a flowchart.

1. **Oval:** It defines start and end of the flowchart and marks it accordingly. Use start to begin the flowchart and end to complete the flowchart.



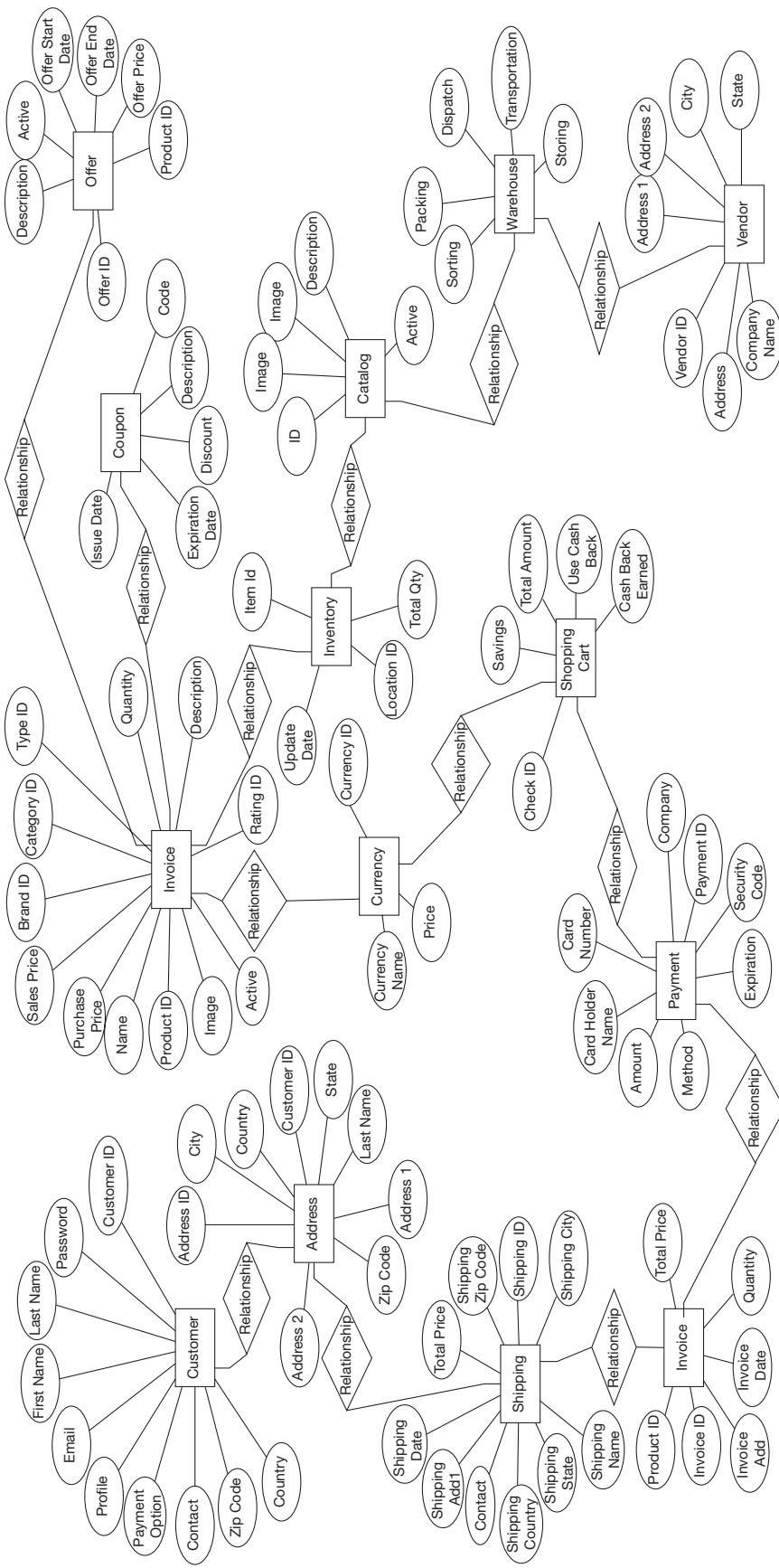


Figure 2.1 Entity relationship diagram of e-commerce entities.

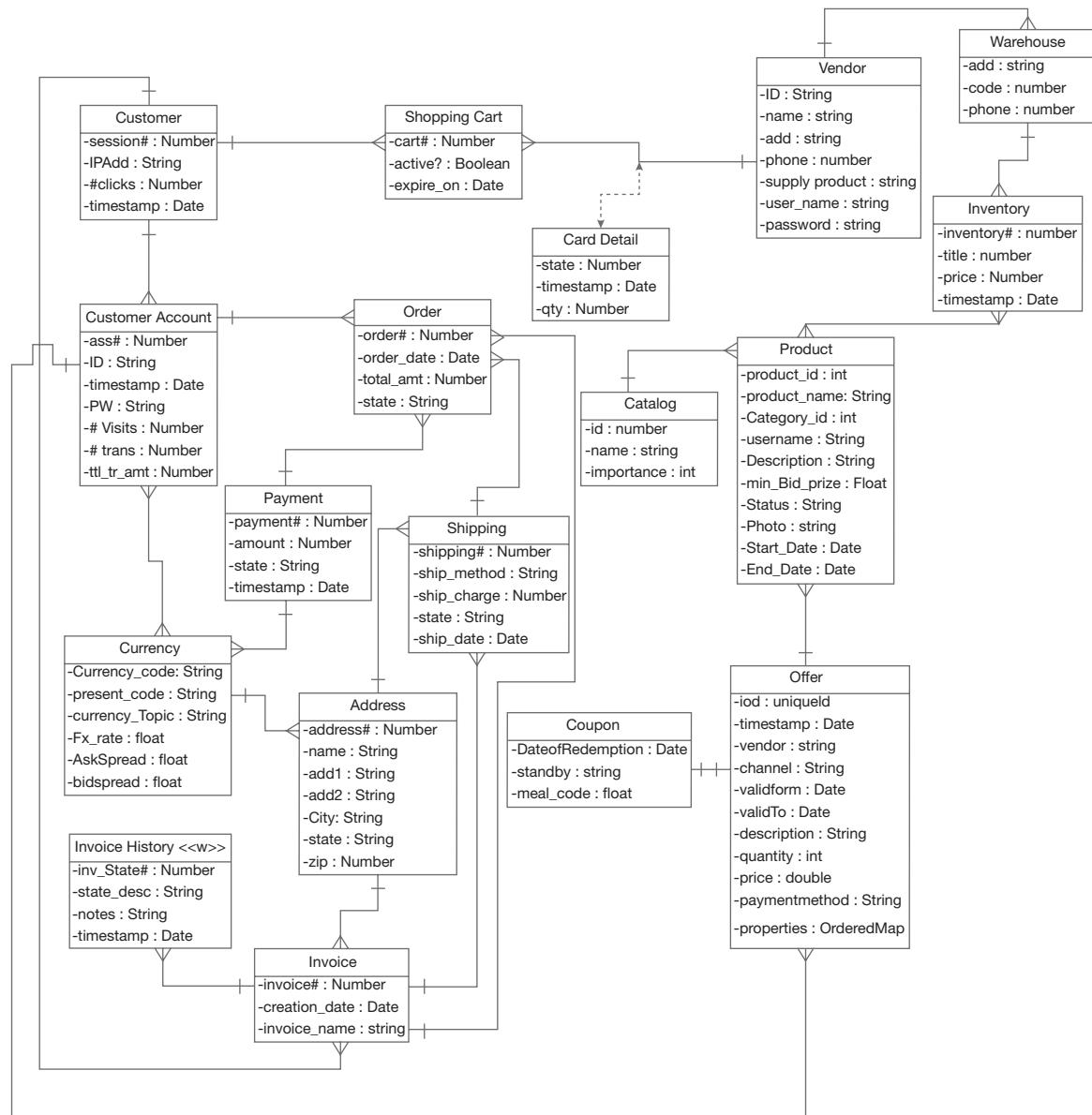
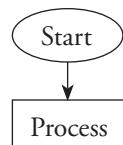


Figure 2.2 UML class diagram of e-commerce classes.

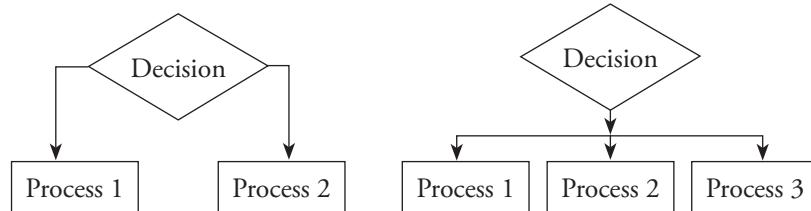
2. **Rectangle:** It is used to define a process step in the flowchart. It is a simple rectangle which contains the name of the process.



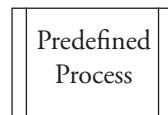
3. **Arrow:** It is used to define a direction of the processing flow. Use this to show how the process data flows in the flowchart.



4. **Diamond:** It is used to specify the decisions to be made in the flow. With this you can show multiple choices as well.



5. **Rectangle with three sections:** It is used to specify predefined process, which is also called *subroutine*. This process is formally defined somewhere else and mainly used for expressing a subprocess.



2.7.1 Login–Registration–Ordering Flowchart

The flowchart in Figure 2.3 shows how the user can login to the front-end website and proceed on completing an online order. It also shows a process where a user can register if not already registered.

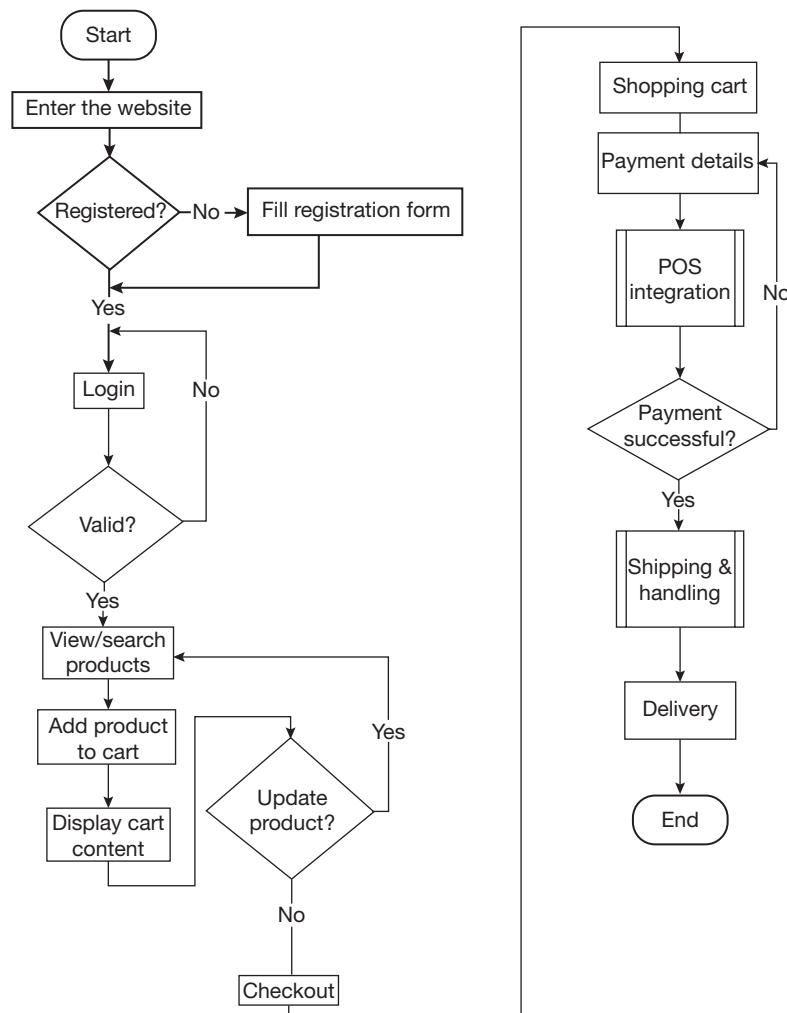


Figure 2.3 Login–registration–ordering flowchart.

Login Registration Process Flow:

- Step 1:** To begin shopping, the customer will need to type the URL or name of the shopping website into a web browser to enter into the website.
- Step 2:** Now, the customer will need a registered account on the website to start shopping. If he/she does not have an account then the website will prompt the customer to create one. The diamond box checks this condition. The website will ask for a valid address, mobile number, name, etc.
- Step 3:** Once the customer has created an account on the website, they need to use their login credentials to proceed further.
- Step 4:** After logging into the website, the customer can start shopping.
- Step 5:** The customer can view different pages of the site or use search options to browse for products to shop.
- Step 6:** Once the customer has selected a product after reading its details and features, they can add it to their cart for further billing.
- Step 7:** The customer needs to click on the show cart items to display cart content.
- Step 8:** Now, the customer needs to decide whether he/she wants to make modifications to the products on his/her cart. For example, changing the product quantity. The diamond box checks this condition.
- Step 8.1:** If the customer has decided to update the product quantity, then he/she can change it by using scrolls. However, if they do not want to update anything, then he/she will proceed directly to the next step, that is, checkout.
- Step 9:** In the checkout window, the customer will be provided with the final summary and pricing of the product he/she is purchasing. If the customer is still not satisfied with the selected products, he/she can go back to Step 4 and repeat the process.
- Step 10:** Next, the customer is directed to the payment options.
- Step 11:** The customer will need to select his/her mode of payment to pay for the products. There will be multiple options for payment.
- Step 12:** The website will guide the customer through the POS integrated checkout system. POS integration helps e-commerce websites to effectively manage sales and purchases from scanning product information and generating receipts after the use of credit/debit card.
- Step 13:** Now, it is up to the bank and vendor to verify and authorize the payment. If it is completed, then it will show a success message and the next step will begin. However, if the payment fails, the customer will be redirected to the payment details page to try again or choose other payment options. The diamond box checks this condition.
- Step 14:** If the payment is successful, the process of shipping and handling begins in which the vendor and a logistic company plays the important role of handling and delivering the product(s) at the doorstep of the customer.
- Step 15:** The product will be delivered to the customer within an estimated delivery time.

2.7.2 Cart Finalization Flowchart

The flowchart in Figure 2.4 shows how a customer can complete ordering the desired products. It takes us through the order finalization stage where the customer can view and update the order if required.

Cart Finalization Process Flow:

- Step 1:** First the website will get information about the cart order.
- Step 2:** Then from the cart order the cart item information will be gathered.
- Step 3:** Then it will redirect the page to the order info page for further processes.
- Step 3.1:** If the customer wants to further update the cart, he/she can still finalize it before posting.
- Step 3.2:** Once the cart items have been finalized by the customer, he/she post cart item(s).
- Step 4.1:** The system calculates the shipping rate for the selected products in the cart and adds it to the final bill.
- Step 4.2:** After finalizing the carts items, the customer can also browse the misc menu option for tiny modifications in the order. For example, the customer can gift wrap the product or schedule the delivery date according to his preference.
- Step 5:** Once Steps 4.1 and 4.2 is complete, the cart items are ready to be posted for billing process.
- Step 6:** The system checks if the entered information is valid or not to complete the next process. To gather the information, the system repeats the Step 2 (get cart item info). However, if the information is not valid, then the process terminates.
- Step 7:** The system calculates the final amount to be paid by the customer.

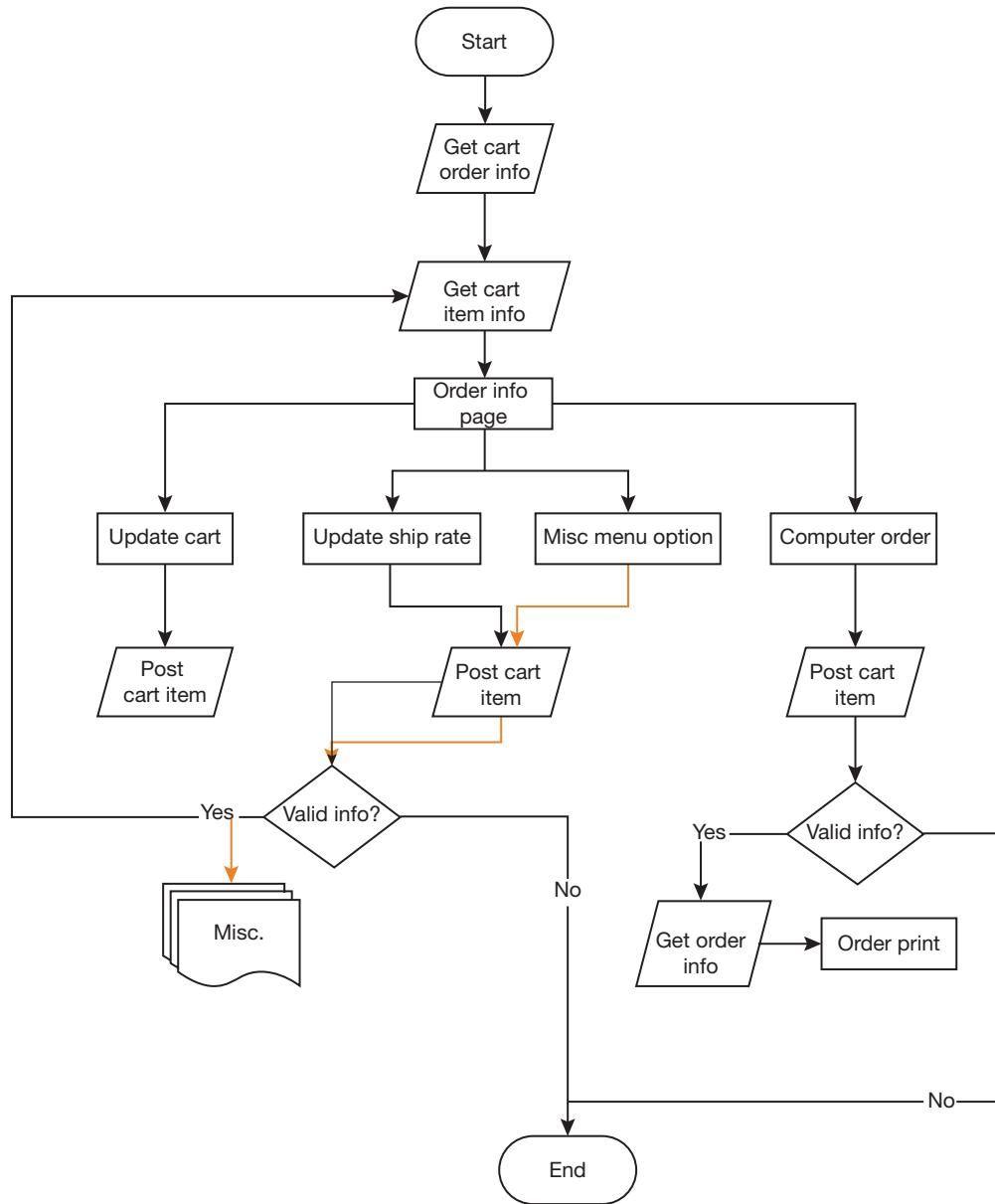


Figure 2.4 Cart finalization flowchart.

- Step 8:** The items in the cart gets posted to create an invoice.
- Step 9:** The system verifies if the information to be printed is valid or not. If the information is not valid, then the process gets terminated.
- Step 10:** If the information is correct, then system gathers order info. This may include the specifications of the product.

2.7.3 Order Processing Flowchart

Figure 2.5 shows the order processing flow. This goes through the back-end process to fulfill the order placed by a customer.

Order Processing Flow:

- Step 1:** First, the customer will place an order on the e-commerce website and the admin will receive a notification.
- Step 2:** Now, the admin will check the details of the order placed by the customer.

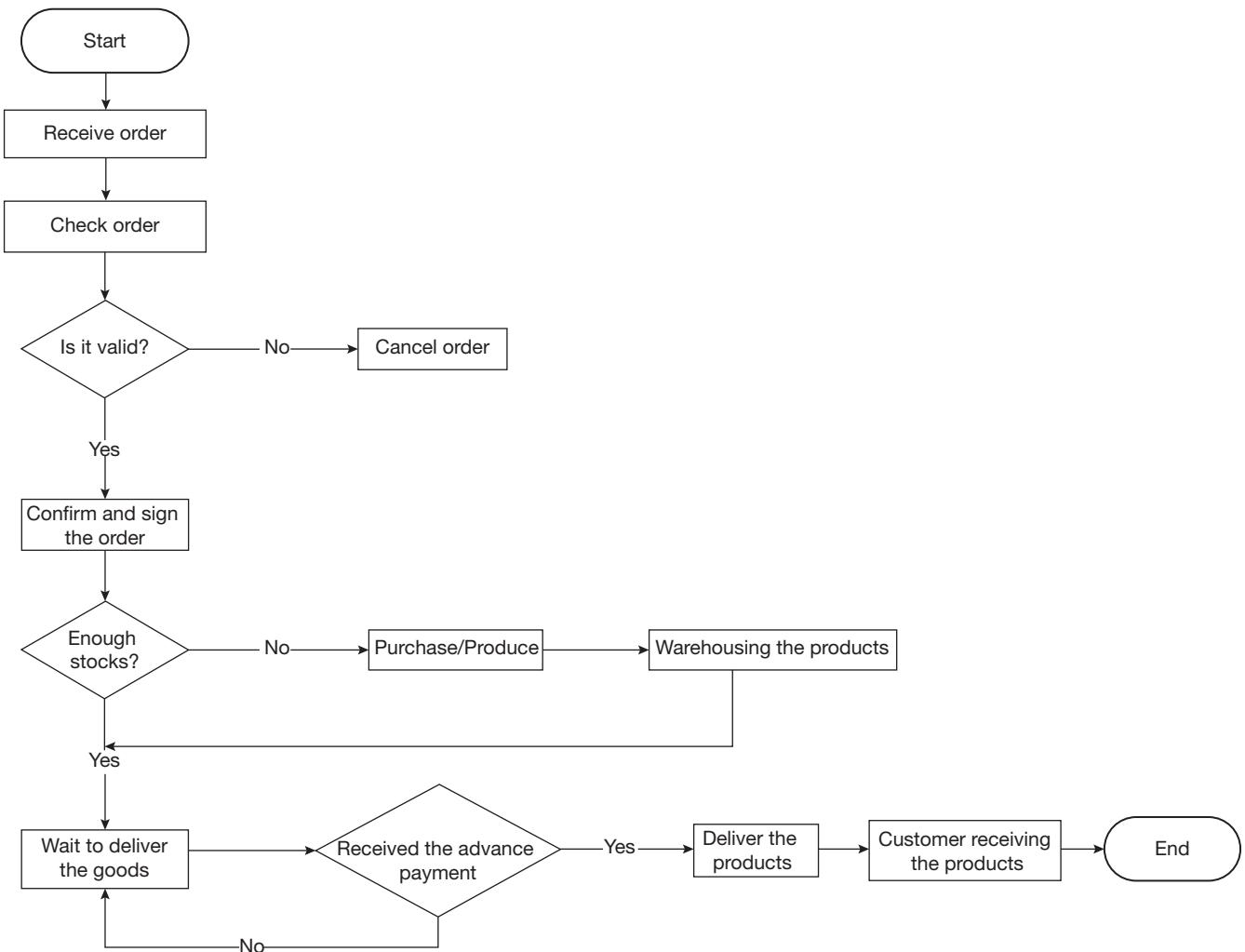


Figure 2.5 Order processing flowchart.

- Step 3:** The admin will determine whether the order is valid or not. For example, e-commerce websites do not let users to stockpile products. In such cases, the order will be cancelled.
- Step 4:** If the order seems to be valid, then the admin confirms and signs the order for further processes.
- Step 5:** Now, the admin needs to check on the stocks to fulfill the order.
- Step 5.1:** If there is insufficient stock in the inventory, then the admin will need to purchase or produce the products from other vendors.
- Step 5.2:** Once the products have been purchased or produced, they need to be added to the warehouse of the e-commerce owner.
- Step 6:** Now, the admin needs to wait to deliver the products before he/she receives payment.
- Step 7:** Advance payment needs to be received before beginning the delivery of the goods. However, if the payment is not received, then the admin will need to wait to dispatch the products.
- Step 8:** The product(s) will be dispatched for delivery once the advance payment has been received.
- Step 9:** At the end, the customer will receive the product within an estimated delivery time.

2.7.4 Vendor Order Processing Flowchart

Figure 2.6 shows how vendor processes the order once received from a customer. It goes through the stock verification process and ends with shipment and payment collection in case of cash on delivery (COD).

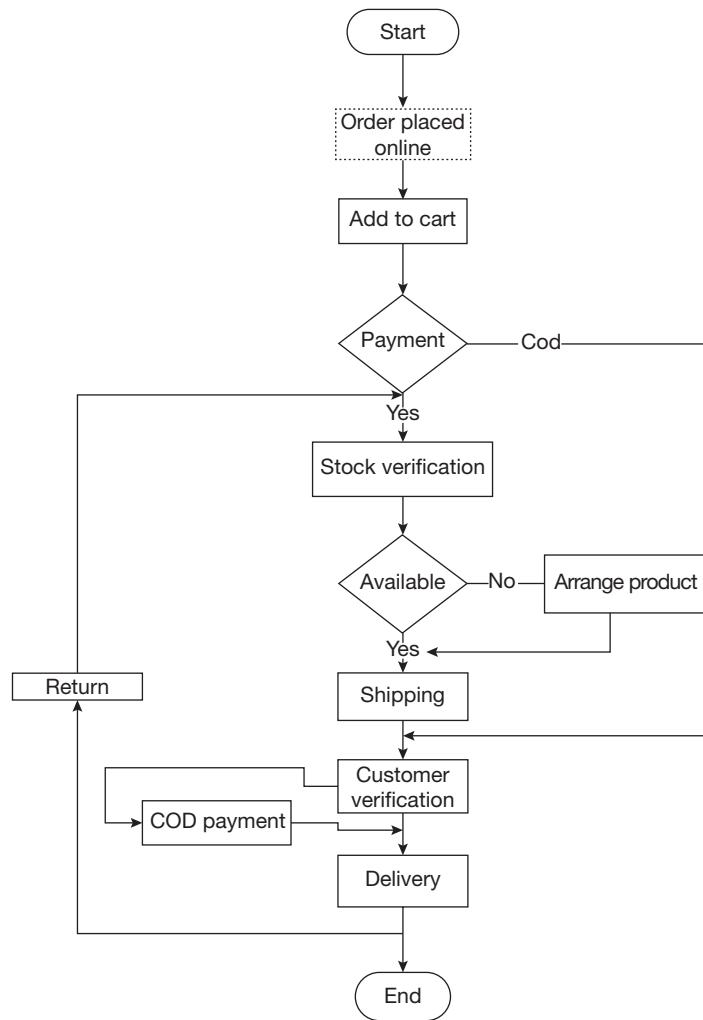


Figure 2.6 Vendor order processing flowchart.

2.7.5 Admin Dashboard Flowchart

Figure 2.7 shows the admin dashboard flow where an admin user can login to the back-end system and perform specific tasks.

Admin Dashboard Process Flow:

- Step 1:** To access the admin dashboard, the admin will need to visit the login page for admins.
- Step 2:** Here the admin is needed to enter a valid username and password.
- Step 3:** If the entered user name and password are correct, then it will direct the admin to the admin dashboard. However, it will redirect the admin to the login page if they have entered the wrong credentials.
- Step 4:** Once the admin has logged in, they will have access to B2B, account manager, and sales report.
- Step 4.1:** In the B2B user tab, admins can manage their connections with other vendors.
- Step 4.2:** Admins can also view their detailed customer information to get informed.
- Step 4.3:** The product catalog will help the admin to manage products on their website. They can add/edit/remove products from there.
- Step 5:** Account Manager Tab lets the admin edit important details. They can tweak website settings from there.
- Step 5.1:** Admins can also browse customer information to tweak the store according to the nature of the customer to improve sales. This will show a list of registered users on their website.

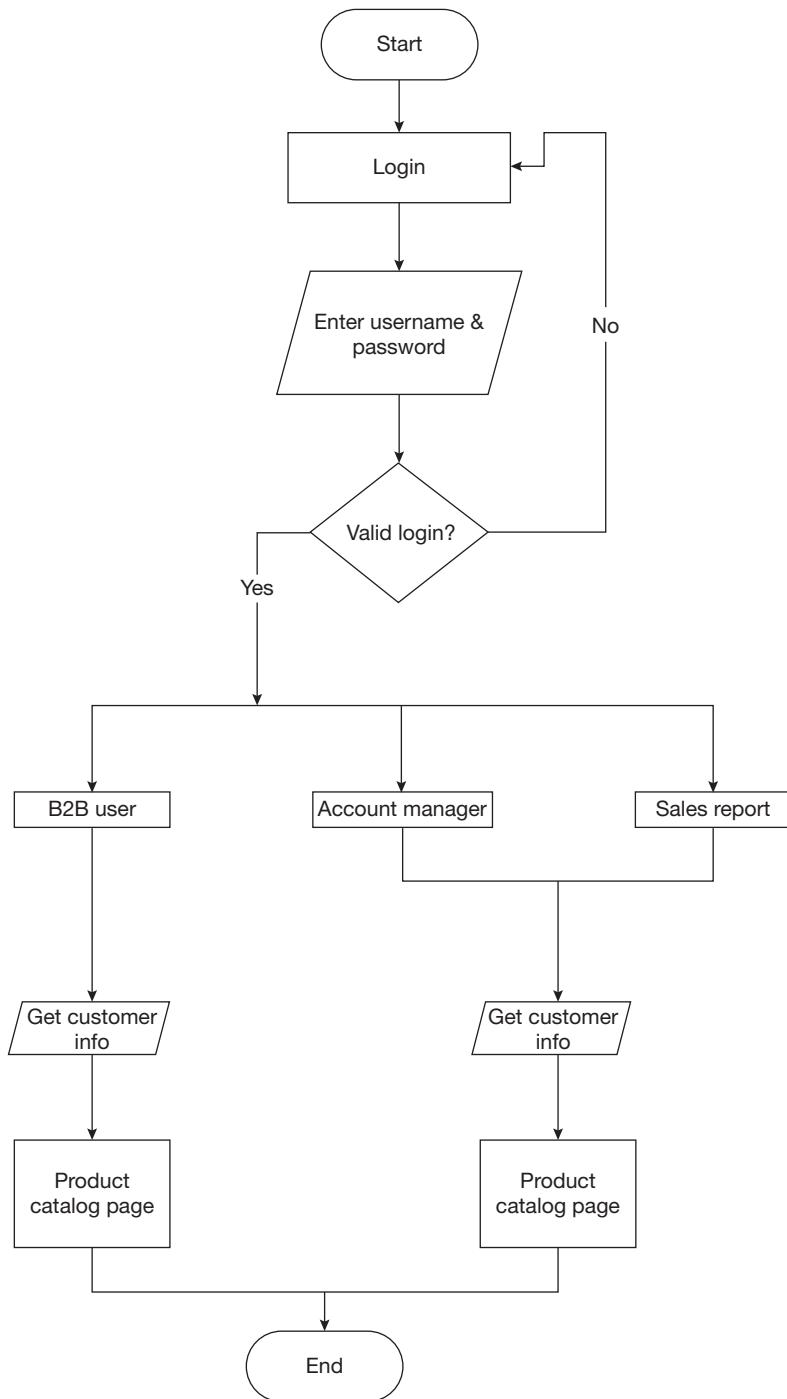


Figure 2.7 Admin dashboard flowchart.

- Step 5.2:** Admins can then browse the product catalog page to tweak the products according to their customer's nature.
- Step 6:** The Sales Report Tab will show a comprehensive report of sales of their products.
- Step 6.1:** The admins can get the customer information through the sales report tab and know their preference and reach.
- Step 6.2:** Then the admins can follow Step 5.2 to tweak the store.

QUICK CHALLENGE

As we have seen in the above examples, flowchart can be made for each major activity in the system. So far, we have covered login, registration, order processing, cart finalization, vendor order processing, and admin dashboard.

Create flowcharts for remaining activities – payment processing, refund and return management, user my account page, user add to cart flow, admin add product process, admin add offer and discount management processes, user delivery tracking process, and admin financials page to show sales. Also, think of adding third-party integration in the flowchart like external chat system, other website integration, Google analytics, affiliate programs, etc. Following are the steps you could take to complete this challenge:

- Identify major activity in the system.
- Identify all the entities related to that activity.
- Figure out the entire flow to complete that activity.
- Consider the conditions and put them in the diamond shape boxes.



2.8 | Front-End Page Flow Design

The front-end page flow diagram in Figure 2.8 shows how the pages communicate with each other and how users interact with the pages. It also shows process flow and the conditions attached to the flow. The flowchart is a login, registration, search, purchase, and payment flowchart.

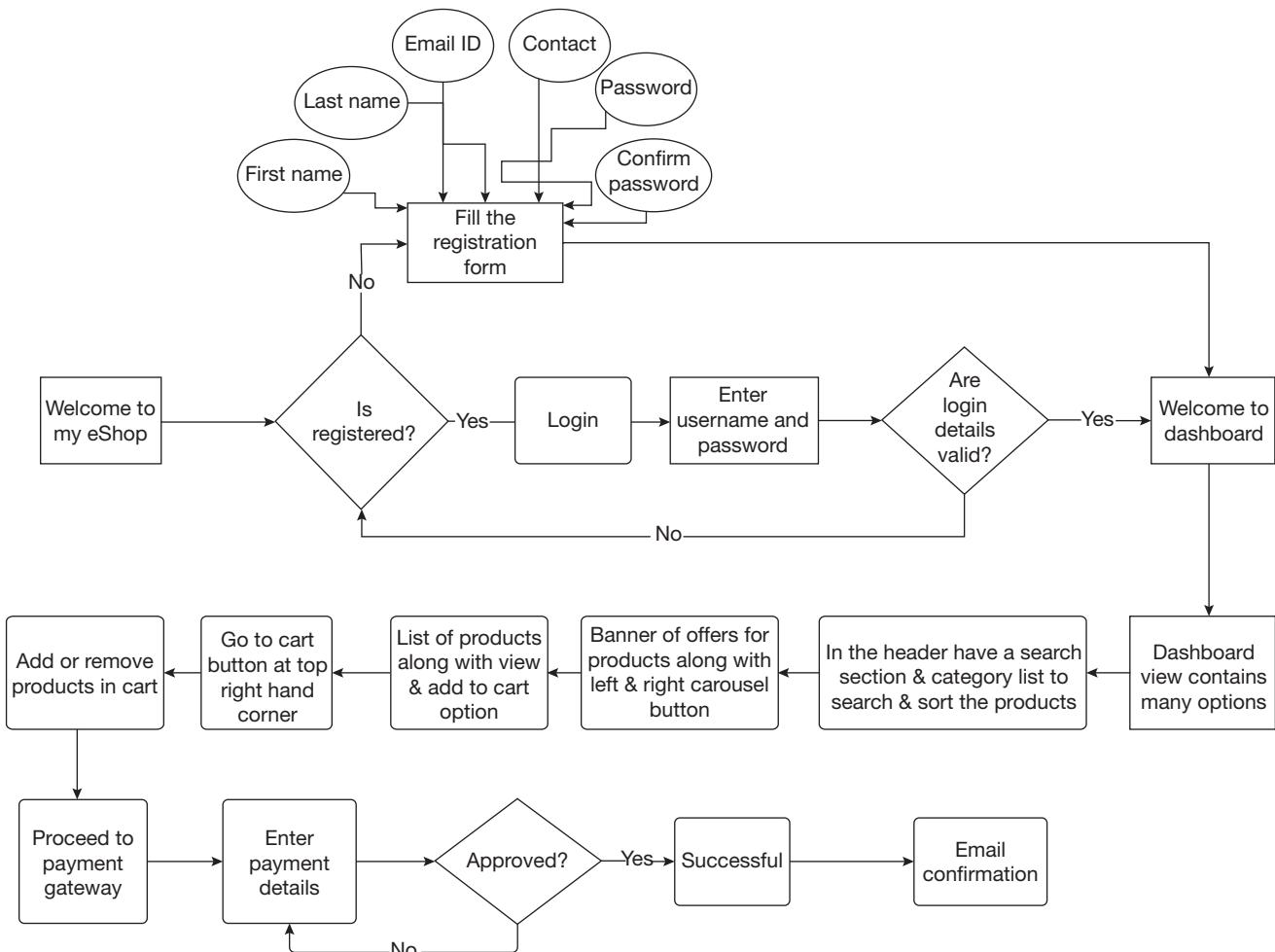


Figure 2.8 Login, registration, search, purchase, and payment flowchart.

**QUICK
CHALLENGE**

Expand this page flow diagram and add the detailed page elements to cover all the possible options and conditions.

2.9 | Back-End Web Services API Endpoints

We need to think of the endpoints that will be needed to get data from the back-end. Think of all the pages we need to feed data to and collect information from to store it in the database.

2.9.1 GET Endpoints

The GET method is used to transfer data from client to server. In this method, data is transferred via URL parameters and hence exposed to the world. Anyone listening to the traffic will see the data, as it is part of the URL. This method should not be used for login, registration, or any sensitive operations as it adds a huge security risk. Also, there is a limitation on sending data via this method as URL length is restricted. Following are some of the examples of GET endpoints.

```
"/customer/getCustomerByEmail/{email}"
```

The above endpoint allows us to fetch customer data using his/her e-mail address. The front-end collects customer e-mail and sends it to the back-end via GET method as mentioned in the endpoint by replacing {email} placeholder. The back-end then returns the customer information for that specific e-mail address in a JSON form. We can parse this JSON response on the front-end side to show the customer information. We have implemented this endpoint in the back-end for you to understand the code.

```
"/customer/getCustomerByID/{id}"
```

The above endpoint allows us to fetch customer data using his/her id. This endpoint works similar to the previous one where we were fetching customer data using his/her e-mail address. There is a slight difference on the back-end though where we will be using “long” type to accept “id” in the parameter list for the controller method compared to “String” in the record fetch via e-mail case.

```
"/order/cancel/{ordernumber}"
```

This endpoint is useful for cancelling the order. Since we need to pass order number only, we can use GET method for this.

```
"/order/status/{ordernumber}"
```

This endpoint is useful to check the status of the order. In this case also we only need the order number and hence GET can be used here.

**QUICK
CHALLENGE**

Based on the ERD, flowcharts and page flow diagrams that we have studied in this chapter, describe other GET endpoints you will need.

2.9.2 POST Endpoints

Following examples show us the use of POST in fetching data from the server. This method is very different from GET as it does not pass data via URL parameters. This method transfers information via HTTP header. The other advantage of POST over GET is that it does not have any limitation on data size so it can be used to transfer large photos, large texts, etc.

```
"/customer/save"
```

This endpoint is very straightforward from the URL point of view as the data is not getting sent via URL. This endpoint is there to create new customer data in the database. All the customer related information is required to pass via header so backend can access it in the same manner.

```
"/customer/update"
```

This endpoint as name suggests is available to update customer information in the database.

"/order/new"

This endpoint is useful to create a new order. Since order data may be sensitive in nature and moreover may contain large information, it is a good idea to send it via POST.

**QUICK
CHALLENGE**

Based on the ERD, flowcharts, and page flow diagrams that we have studied in this chapter, describe other POST endpoints you will need.

2.9.3 GET versus POST



In this section we will discuss the differences between GET and POST methods. As you have learnt, both methods are used to send data from client to server. However, they have a different working pattern, as listed in Table 2.1.

Table 2.1 Differences between GET and POST methods

	GET	POST
History	Parameters are passed as part of the URL and hence are stored in browser history.	Parameters are passed via HTTP header and hence do not get saved in browser history.
Bookmarked	As it is part of URL, it can be bookmarked.	Parameters cannot be bookmarked as header information does not get bookmarked.
Cached	GET is cacheable.	POST is not cacheable.
Re-submit/ Re-execute behavior	In case GET is used in the HTML form and gets cached, it cannot be resubmitted but it can be re-executed.	POST can be re-submitted but browser will alert user about it.
Encoding Type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data
Parameters limitation	URL length has some limitation on sending data.	No restrictions on data size.
Restrictions on form data type	In this mode, only ASCII characters are allowed.	There is no restriction on this and any type of data can be transmitted.
Hack severity level	Visible to public so it is less secure.	Data is sent via HTTP headers and in case of SSL, data is encrypted and hence not easy to hack.
Security	Since data can be stored in the history or bookmarked, it is available in a plain text and may expose to outside world.	This is not the case with POST so the data is not available to the outside world, making it more secure than GET.
Usability	Useful for only handful of small operations like fetching a record with ID or other parameters, etc.	Useful for sending any type of data including username/password, bank information, etc.



Describe at least 10 advanced endpoints like predictive analysis for product selection, behavior tracking for buying choices, buying pattern for offers and promotions, etc.

There are many ways through which we can send data from client to server. However, only GET and POST are popular. The other methods are as follows:

- PUT:** This method is similar to POST method, but one major difference is PUT requests are idempotent which means calling the same PUT request will produce the same result. However POST will post data again and again, forcing the back-end to create multiple records.

2. **HEAD:** This method is similar to GET but one difference is it does have response body. For example, if GET method is fired on the customer controller with a parameter such as ID, server will return the customer data for that ID. However, in the case of HEAD, the data will not get returned as the response body is missing.
3. **DELETE:** This method is useful to delete a specified resource.
4. **PATCH:** This method is useful for the partial modification to a resource and does not allow complete replacement. This method is not idempotent like POST, so it will affect the data.
5. **OPTIONS:** This method describes the communication options for the resource.

Summary

As we have seen, application development takes a lot more effort than simply writing the code in a favorite language. There are various steps involved in designing and developing an application. Planning phase is equally important as it gives a clear blueprint of what needs to be developed. In this chapter, we have walked through the entire process and are now in a position to start working on our application. The next step is to gain the skills required to start the development process and develop further in the upcoming chapters. In this chapter, we have learned the following concepts:

1. Basic project idea, complexity, and implementation strategy.
2. Various architectural diagrams like flowchart, page flow, entity relationship diagram, etc.
3. Design thinking process and identify entities and classes.
4. Flow and relationships between entities and classes.
5. Web services API endpoints and their uses.

In the Chapter 3, we will learn about HTML, which is dominantly used in the industry to create front-end web pages. We will explore the basic building blocks of HTML and various tags, elements, and attributes. We will see some practical examples which will give you a clear idea on using those in a real-life project.

Multiple-Choice Questions

1. While creating Entity Relationship diagram, what is denoted by double ovals?
 - (a) Multi-value key
 - (b) Multi-value entity
 - (c) Multi-value table
 - (d) Multi-value attributes
2. Which of the following facilitates graphical representation of a given problem?
 - (a) Pseudocode
 - (b) Algorithm
 - (c) Flowchart
 - (d) All of the above
3. Which one of the following symbols is utilized at the beginning of a flowchart?
 - (a) Rectangle
4. What is the main benefit of a User Flow Diagram?
 - (a) Spot potential problems
 - (b) Remove virus
 - (c) Draw comparison between two things
 - (d) Hide files
5. Derived attributes are denoted by _____ in entity relationship diagrams.
 - (a) Dotted Square
 - (b) Dotted Rectangle
 - (c) Dotted Triangle
 - (d) Dotted Oval

Review Questions

1. What is flowchart?
2. What is UML Class Diagram and what is its use?
3. In how many ways are entities related to each other?
4. Why is entity relationship diagram essential?
5. How can one identify dependencies between entities?
6. What is the benefit of using POST method over GET?
7. When will you use PUT over POST method?
8. What is PATCH and when should one use it?
9. What is the benefit of using DELETE method over POST to delete a record?

Exercises

1. Create a flowchart for booking an airline ticket.
2. Define entities for airline booking web application.
3. Decide on classes that you need based on the entities and create a class diagram.
4. Draw page diagrams for the complete search and booking process.
5. Create a detailed comparison chart for POST, GET, PUT, HEAD, DELETE, PATCH, and OPTIONS.
6. Explain the benefits of architectural diagrams.

Project Idea

Study the following requirements of a taxi-booking web application and design a complete architectural diagram for the same. Explain all the entities and draw an entity relationship diagram. Based on these entities draw class diagrams and identify the relationships between these classes. Define a flow of the application and design page flow diagrams. Identify web service APIs that front-end will be calling and come up with data transfer methods for the API methods.

Taxi Booking Application: In an urban setup, it is extremely difficult to book a taxi as the only option for an

individual is to wait for a passing by taxi and check if it is empty or not. In many cases, empty taxis are not available and the user ends up being canceling his trip or uses other mode of transport or simply walks if the destination is nearby. This problem also affects taxi drivers, as they do not get enough commuters to make a sustainable living. Having a taxi booking application where user can look for nearby taxies that are available and book one of them can solve this problem.

Recommended Readings

1. Axel Van Lamsweerde. 2009. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley: New Jersey
2. Hans Van Vliet. 2010. *Software Engineering: Principles and Practice, Third edition*. Wiley: New Jersey
3. Rajib Mall. 2018. *Fundamentals of Software Engineering, Fifth edition*. PHI Learning: New Delhi
4. Pankaj Jalote. 2010. *Pankaj Jalote's Software Engineering: A Precise Approach*. Wiley: New Delhi
5. Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. 2013. *Database System Concepts, Sixth edition*. McGraw Hill Education: New York

Introduction to Hyper Text Markup Language

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- HTML and its importance in web development.
- Structure of HTML code.
- Basic building blocks of HTML and its syntax.
- Various elements needed to build and deploy HTML webpage.
- How to design a webpage using HTML.

3.1 | Overview of HTML



HTML stands for HyperText Markup Language. It is one of the primary front-end technologies used for front-end development. HTML is irreplaceable, that is, no other technology can replicate its purpose. The purpose of HTML is to describe the structure of a webpage. Shown below is a simple HTML example.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First HTML Example</title>
  </head>
  <body>
    <h1>Let us Learn HTML History</h1>
    <p>HTML is the most popular language for webpages that was created in early
1990s</p>
  </body>
</html>
```

Now, let us dissect the structure of HTML code. Here,

1. `<!DOCTYPE html>` indicates the document to be treated is HTML 5.
2. `<html>` tag is the root element of our page.
3. `<head>` carries meta information.
4. `<title>` tag is used to define a title and appears written at the top of the page in the browser tab.
5. `<body>` carries the complete content of the page where `<h1>` is used for the first heading and `<p>` is used for a paragraph.

Is this too much information at once? Well, do not worry. We are going to closely examine each of these components and go deeper into HTML. Figure 3.1 shows the element hierarchy.

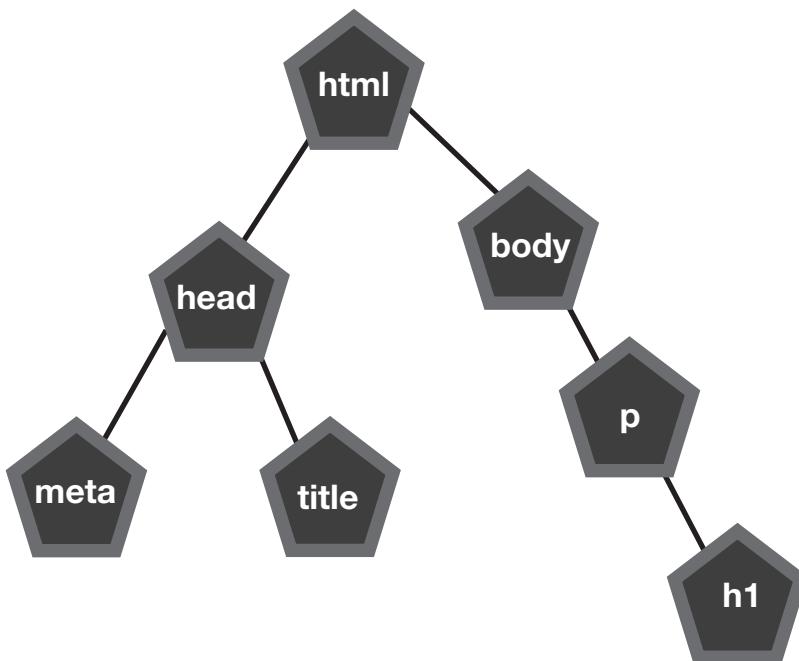


Figure 3.1 Element hierarchy.



Why should we use `<!DOCTYPE>` element? What will happen if we do not use it?

3.2 | Getting Started with HTML Code

Before going over the examples of HTML, you would require a text editor. While production environments use text editors like VS Code, Sublime or Atom, for beginners Notepad is good enough to work for Windows users andTextEdit does the job for Mac users. To write in HTML, follow these steps:

1. **Open notepad:** Open start and enter “Notepad”. A blue color icon with “notepad” would show, open it.
2. **Type:** Type “HTML”

Use the example in Section 3.1 and type it in your Notepad.

```

<!DOCTYPE html>
<html>
  <head>
    <title>My First HTML Example</title>
  </head>
  <body>
    <h1>Let us Learn HTML History</h1>
    <p>HTML is the most popular language for webpages that was created in early
1990s</p>
  </body>
</html>
  
```

3. **Save:** Now go to File → Save As and type “firstexample.html”. The extension automatically converts your file into an HTML one.
4. **Check yourself:** After saving, go to your “.html” file in the folder and open it in any of your browsers by right-clicking and selecting “Open With”. In response, you can see your first webpage as shown in the following screenshot.

Lets Learn HTML History

HTML is the most popular language for web pages that was created in early 1990s

QUICK CHALLENGE

Find out all the browsers available on the Internet to download and list the differences.



3.3 | Important Components of HTML

HTML is made up of various components. In this section, we will discuss all the parts which make an HTML document.

3.3.1 Tags

HTML contains a certain feature known as tags. The format of a tag is

```
<tagname> any content </tagname>
```

Mostly, tags occur in a pair. The opening tag is also known as *start tag* while the closing tag is known as *ending tag*. Before the tag name of the closing tag, there is a forward slash.

Internet browsers such as Chrome, Firefox, and Internet Explorer go over documents written in HTML and present an output by displaying it. The browser processes tags and calculates the content which should be displayed to the users.



Which international standards body publishes HTML specifications?

3.3.2 Elements

The whole line consisting of tags, including attributes and the content inside them, is referred to as *element*. Without content, an element is known as *empty element*. As you may have noticed in the above example, HTML elements support nesting which means an element can accept another element in it. For example, `<html>` entails other HTML elements. The following are a few examples of elements:

- `<p>` : This element is useful to create a paragraph.
- `<a>` : This element is useful to create a hyper link.
- `<h1>` : This element is useful to create a first heading.
- `<h2>` : This element is useful to create a second heading.
- `<h3>` : This element is useful to create a third heading.
- `<h4>` : This element is useful to create a fourth heading.
- `<div>` : This element is useful to create a division or section.
- `` : This element is useful to make the given text bold.
- `
` : This element is useful to create a single line break.
- `<button>` : This element is useful to create a button.

3.3.3 Attributes

To represent further information about how an element should be displayed, attributes are used. Each element can have an attribute. They are always defined in the opening tag. They are written in the following format:

```
name="value"
```

QUICK CHALLENGE

What is the fault in the following HTML markup?

```
<!DOCTYPE html>
<html>
  <body>
    <head>
      <h1>Hello World</h1>
    </head>
  </body>
</html>
```

3.3.4 Global Attributes

The following are the global attributes of HTML5, which can be used with any tag.

accesskey	hidden	itemtype
class	id	lang
contenteditable	insert	spellcheck
contextmenu	itemid	style
dir	itemprop	tabindex
draggable	itemref	title
Dropzone	itemscope	Translate

Table 3.1 provides the explanation of these attributes.

Table 3.1 Attributes and their explanation

Attribute	Explanation
accesskey	This attribute helps us to know that a shortcut key can be utilized to explore the element. (Any string of characters. This string of characters helps us to acknowledge the key/s the user requires to utilize in order to explore the element.)
class	This attribute is known as document-wide identifier. It is used to assign a class stated in the style sheet. The value could be the specific name of the class you desire to use.
contenteditable	This attribute lets the user know whether he/she has the authority to edit the content or not. Possible values under this attribute are <ul style="list-style-type: none"> True False
contextmenu	This attribute helps to define a context menu for an element. In this, the value is bound to be the ID of a menu element present in the Document Object Model (DOM). DOM is an application programming interface for HTML, XHTML, and XML. DOM specifies the elements in a tree structure wherein each node is considered as object.
dir	This attribute is used to set direction of the text. Also, please note that Dir is not supported in HTML5. You are advised to use CSS instead. Possible values in this attribute are: <ul style="list-style-type: none"> Ltr: Defines that the text must be read left to right. Rtl: Defines that the text must be read right to left. Auto: This is a rather unique attribute; the direction of the text must be defined programmatically by using the contents present in the element.
draggable	This attribute defines whether the element is allowed to be dragged or not by the user. Possible values in this attribute are: <ul style="list-style-type: none"> True: This confirms that the element is draggable by the user. False: This confirms that the element is not draggable by the user. Auto: This utilizes the default action of the user agent/browser. The value is the default one.

(Continued)

Table 3.1 (Continued)

Attribute	Explanation
dropzone	This defines what would happen when a user “drops” an element onto another current element. This must consist of an unordered set of unique space-separated tokens that are ASCII and case-sensitive. Possible values in this attribute are: <ul style="list-style-type: none"> • Copy: Provides a copy of the dragged data. This is the default value. • Move: Moves the data into a new location. • Link: Creates a link to the original data.
hidden	Helps us to know whether the element is no longer or not yet relevant. Thus, the browser or user agent will not display those elements to the user which have this hidden attribute in them. However, this is a Boolean attribute. This means that if the attribute is present somewhere, then its value must be an empty string or a unique value with an ASCII case-insensitive character to fill the gap. Thus, you cannot leave a trailing white space. Possible values in this attribute are: <ul style="list-style-type: none"> • (Empty String) • Hidden
id	This is a document-wide identifier as it is used with CSS and JavaScript. The value of this attribute could be the name of the id you desire to use.
inert	This is a Boolean attribute which defines that the element is meant to be inert. However, the browser or the user agent might behave that the element is not present, even though it may still display the entire element. Example: A block of plain text that has been highlighted with the inert attribute may not be discoverable when a user tries to find it in the text search present in the browser. This concludes that the text may not be meant for the users to interact with. Thus, the user will not be able to select the text block. As it comes under Boolean attribute, it means that if the attribute is present somewhere, then its value must be an empty string or a unique value with an ASCII case-insensitive character to fill the gap. Thus, you cannot leave a trailing white space. Possible values in this attribute: <ul style="list-style-type: none"> • (Empty String) • Insert
itemid	This is an optional attribute which provides a global identifier for an “item”. If provided, the value must be a valid URL enclosed by spaces. This attribute can only be specified in elements which consist both itemscope and itemtype attributes, considering the vocabulary used by itemtype adheres to the global identifiers for items vocabulary.
itemprop	This attribute gives one or more properties to one or more than one “items”. However, this attribute remains optional and it could be utilized as any HTML element. If it is utilized, then it must consist of a value that works as an unordered set of unique space-separated tokens that are case-sensitive in nature. Plus, they should be representing the names of the name-value pairs that they add. Thus, the itemprop attribute must have at least one token. If we follow the HTML5 specifications, then each token must be one of the following: <ul style="list-style-type: none"> • A URL that is absolute but valid URL. • If the item is a written item: a “defined property name” permitted in this situation following the specifications that define the similar types for the item. • If the item is not a written item: a string that consists no U+002E FULL STOP characters (.), no U+003ACCOLON characters (:), and no spaced characters.
itemref	Utilized in conjunction along with the itemscope attribute. This attribute gives us a list of extra elements to find the name-value pairs of the “item”. However, the itemref attribute remains optional. In case of its usage, it must consist of a value that is an unordered set of unique space-separated tokens that too is case-sensitive. Plus, it must be having IDs of elements in the same home subtree. Lastly, the itemref attribute can only be used by a user on elements having the itemscope attribute present in them.
itemscope	HTML5 elements that consist of itemscope attribute generate a name-value pair known as “item”. Elements associated with an itemscope attribute might also consist of an itemtype attribute specified. This is done to provide the item types of the item. This is also Boolean attribute, which means that if the attribute is present somewhere, then its value must be an empty string or a unique value with an ASCII case-insensitive character to fill the gap. Thus, you cannot leave a trailing white space. Possible values in this attribute are: <ul style="list-style-type: none"> • (Empty String) • itemscope

(Continued)

Table 3.1 (Continued)

Attribute	Explanation
itemtype	This attribute gives the user an item type for the elements consisting of itemscope attribute. However, itemtype attribute is optional. In case it is used by the user, then it must consist of a value that is an unordered set of unique space-separated tokens that too is case-sensitive, each of which should be valid and absolute URL. Plus, each of which must be assigned to utilize the same vocabulary. At least one token must be present in this attribute's value. <i>Note:</i> The itemtype attribute can only be present in the elements that include the itemscope attribute.
Lang	This attribute helps the user to define the language code to be utilized. Possible values in this attribute are: <ul style="list-style-type: none"> • Proper valid RFC 3066 language code • (Empty string)
spellcheck	It helps to define whether the spelling of the element should be checked or not. Possible values are: <ul style="list-style-type: none"> • (Empty string): The element must have its spelling checked. • True: The element must have its spelling checked. • False: The element doesn't need to have its spelling checked. If the spellcheck attribute is not present, then the element will utilize the default behavior. It might be based on the parent's own spellcheck position.
style	It helps the user to define the inline styles for the element. The value, in this case, should be style definition you desire to use.
tabindex	It helps the user to establish the tabbing order. (It is when the user “tabs” via the elements on the page.) Possible values of this attribute: It could be any integer, example, 0, 1, 2, 3, 4, 5, 6, 7, etc.
title	It defines a specific title to assign with the element. Various browsers might display this when the user hovers the cursor over the element. Possible values for this attribute: Any text that is meant to be displayed as a “tool tip”.
translate	It helps the user to define whether the element's attribute values and the other values of its text node children are needed to be translated when the page gets localized, or whether to dispose them untouched. The translate attribute is an important attribute and it might contain the following possible values: <ul style="list-style-type: none"> • (Empty String) • Yes • No If the user is provided with the translate attribute, but its value is missing or is stated as invalid, then the element will automatically inherit its value from its parent element.

3.3.5 Event Handler Content Attributes

These attributes empower HTML tags to invoke a script when a certain “event” occurs. When a user visits the website, he/she will be doing a lot of things such as clicking on things, hover over elements, etc. These are some common examples of what JavaScript specifies as events. In your HTML code, you can add an event handler attribute to an HTML element to capture an event for a specific user action. For example, you can add the **onMouseover** event handler attribute to a button and assign some JavaScript to explore whenever the user hovers over that button.

Content attributes of event handler are often regarded as “event handlers”. Below are the event handlers that are supported by HTML5. Table 3.2 will help you to know about the event handlers that can be utilized in all HTML5 elements as both IDL attributes and content attributes, and on Window and Document objects.

Table 3.2 Event handlers that can be utilized in all HTML5 elements

Attribute	Explanation
onabort	It occurs when an event has been cancelled or aborted. Example, the web browser stops fetching media data before downloading completes.
oncancel	It allows to specify code to get executed when a cancel event occurs.

(Continued)

Table 3.2 (Continued)

Attribute	Explanation
oncanplay	This attribute is requested when the browser/user agent has the permission to start playing a media, but unable to do so due to buffering. In other words, this means if you start the playback of media, it would not be able to play it completely. This is due to the current playback rate. As a result, you will need to pause during the playback.
oncanplaythrough	The web browser/user agent assess that if the playback is meant to begin now, then the media source could be processed at the present playback rate towards its end and without getting stopped for more buffering.
onchange	It means that the user has altered the object, and then tries to leave that field (i.e., by clicking somewhere else).
onclick	It occurs when the user clicked on the object.
onclose	It occurs when a window is closed.
oncontextmenu	It occurs when the user has triggered a context menu.
oncuechange	It is used to specify a script to run when the <track> element has cue changes.
ondblclick	It occurs when the object has been clicked twice by the user.
ondrag	It occurs when an element has been dragged by the user.
ondragend	It occurs when the user has stopped dragging an element.
ondragenter	It occurs when the user has dragged an element to a drop target.
ondragexit	It occurs when drag target is exited by the element.
ondragleave	It occurs when a valid drop target is left by an element.
ondragover	It occurs when the user drags an element over a valid drop target.
ondragstart	It occurs when a drag task has begun.
ondrop	It occurs when an element has been dropped.
ondurationchange	It occurs when the length of the media has been changed.
onemptied	It occurs when the media source comes out to be empty. This could be a result of network failure.
onended	It occurs when the media file has arrived at its end.
oninput	It occurs when user input is provided.
oninvalid	It occurs when submittable input element is invalid.
onkeydown	It occurs when a key has been pressed above an element.
onkeypress	It occurs when a key has been pressed above an element and then gets released.
onkeyup	It occurs when a key has been released above an element.
onloadeddata	It occurs when the browser/user agent is able to process the media data at the present playback position for the very first time.
onloadedmetadata	It occurs when the browser/user agent has estimated the time and dimensions of the media file.
onloadstart	It occurs when the browser/user agent has begun processing the media source.
onmousedown	It occurs when the cursor has been moved over the object, while the mouse/pointing device was pressed down.
onmousemove	It occurs when the cursor has been moved while it was hovering over an object.
onmouseout	It occurs when the cursor has been moved away from the object.
onmouseover	It occurs when the cursor has been moved over the object. This happens when the user has hovered the mouse over the object.
onmouseup	It occurs when the mouse has been released after being pressed down.
onmousewheel	It occurs when the user has rotated the mouse wheel.
onpause	It occurs when the user has paused the media source.
onplay	It occurs when the user has started the media resource's playback.
onplaying	It occurs when playback has started.

(Continued)

Table 3.2 (Continued)

Attribute	Explanation
onprogress	It occurs when the browser/user agent is trying to fetch data.
onratechange	It occurs when the playback rate has been changed. This happens when either the defaultPlaybackRate or the playbackRate has been updated from its source.
onreset	It occurs when the form is reset.
onseeked	It starts when the seeking IDL attribute has been turned into false. This happens when the seeking attribute is no longer true.
onseeking	It occurs when the seeking IDL attribute has been changed into true and the seek operation is taking too much time and the user agent has enough time to throw the event.
onselect	It occurs when few or all the contents of an object have been selected by the user. For example, when the user has selected some text inside a text field.
onshow	It occurs when menu element is shown.
onstalled	It occurs when the browser or the user agent is trying to fetch some media data but the data is unable to cooperate. This happens when the data has stopped coming to the browser/user agent.
onsubmit	It occurs when the user has submitted a form.
onsuspend	It begins when the browser/user agent is unable to fetch media data and has not downloaded the entire media file (i.e., it has terminated downloading the data).
ontimeupdate	It occurs when the user has changed the current media playback position.
onvolumechange	It occurs when the user has changed the volume or muted attribute.
onwaiting	It occurs when the frame for the media is not available yet, but either the browser or the user agent is expecting it to be available.

3.3.5.1 Event Handlers with Specific Rule

Table 3.3 shows the event handlers that must be backed by all HTML elements other than the **frameset** and **body** elements.

Table 3.3 Event handlers that must be backed by HTML elements other than **frameset** and **body** elements.

Attribute	Explanation
onblur	The element has lost the focus.
oneerror	It occurs when an error takes place while the window object is getting loaded. It is also a handler for the script error notifications.
onfocus	It occurs when the element has focus.
onload	It occurs when the element has been loaded.
onscroll	It occurs when the user has scrolled the element's scrollbar.

Table 3.4 shows the event handlers that must be supported by window objects.

Table 3.4 Event handlers that must be supported by window objects.

Attribute	Explanation
onafterprint	It occurs when printing has started or print dialog is closed.
onbeforeprint	It occurs when page is going to get printed.
onbeforeunload	It occurs when document is about to be unloaded.
onblur	The element has lost the focus.
onerror	It occurs when error takes place while it was loading an element. It is also a handler for script error notifications.
onfocus	It occurs when the element is having the focus.
onhashchange	It gets executed when the user navigates to a session history entry whose URL is different from the previous one, only inside the fragment identifier.
onmessage	It occurs when a message is received from an event source.
offline	It occurs when browser is offline.

(Continued)

Table 3.4 (Continued)

Attribute	Explanation
ononline	It occurs when browser is online.
onpagehide	It occurs when user navigates away from the page.
onpageshow	It occurs when user navigates in to the page.
onpopstate	It occurs when user navigates between history states that a developer has set.
onresize	It occurs when the window has been resized.
onscroll	It occurs when the user scrolls an element from the scrollbar.
onstorage	It occurs when the web storage area changes in context to other document.
onunload	It occurs when page has been unloaded.

3.3.5.2 Event Handlers that are Required for Document Objects Only

The following event handler can be utilized on **Document** objects as IDL attribute:

onreadystatechange – It occurs when the readyState property changes.



Which HTML5 elements begin with K?

3.3.6 Headings

Headings in HTML can be represented by six tags, from *h1* to *h6*. The lower the number of a heading is, the more important and bigger it is. For instance,

```
<!DOCTYPE html>
<html>
  <body>
    <h1>This is our first heading</h1>
    <h2>This is our second heading</h2>
    <h3>This is our third heading</h3>
    <h4>This is our fourth heading</h4>
    <h5>This is our fifth heading</h5>
    <h6>This is our sixth heading</h6>
  </body>
</html>
```

The preceding code shows the following result in a browser window.

The screenshot shows a web browser window with a title bar displaying the URL "file:///Users/MayurRamgir/Documents/headings.html". The main content area of the browser displays the following text structure:

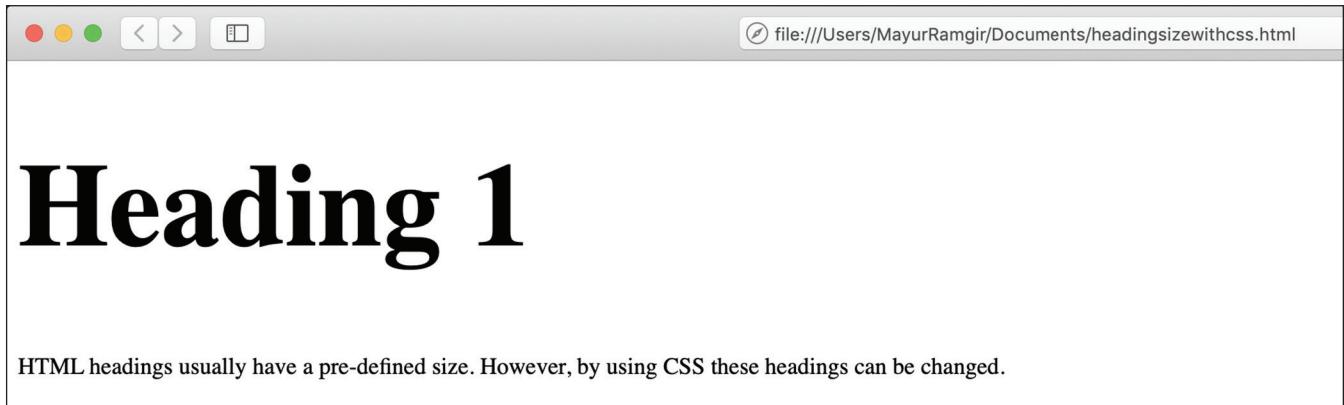
```
This is our first heading
This is our second heading
This is our third heading
This is our fourth heading
This is our fifth heading
This is our sixth heading
```

As you can see for yourself, headings size became smaller as the heading number increased. Often *h1* heading is used to add keywords for better search engine optimization. Additionally, it is necessary for readability because many users use them as a measure to get an idea about the whole content of the website.

The size of a HTML heading is pre-defined but they can be altered by using CSS. For instance,

```
<!DOCTYPE html>
<html>
  <body>
    <h1 style="font-size: 80px;">Heading 1</h1>
    <p>HTML headings usually have a pre-defined size. However, by using
       CSS these headings can be changed.</p>
  </body>
</html>
```

The preceding code shows the following result in a browser window.



For putting a horizontal rule or thematic break between HTML elements, the *
* tag is used. For example,

```
<!DOCTYPE html>
<html>
  <body>
    <h1>1st Heading</h1>
    <p>After this the first break comes</p>
    <hr>

    <h2>2nd Heading</h2>
    <p>After this the second break comes</p>
    <hr>

    <h3>3rd Heading</h3>
    <p>After no break comes</p>

  </body>
</html>
```

The preceding code shows the following result in a browser window.

After this the first break comes

2nd Heading

After this the second break comes

3rd Heading

After no break comes



Will search engines like Google levy penalties which result in downgrading your website's rankings or completely remove it from search engines' indexed database for not using an *<h1>* tag?

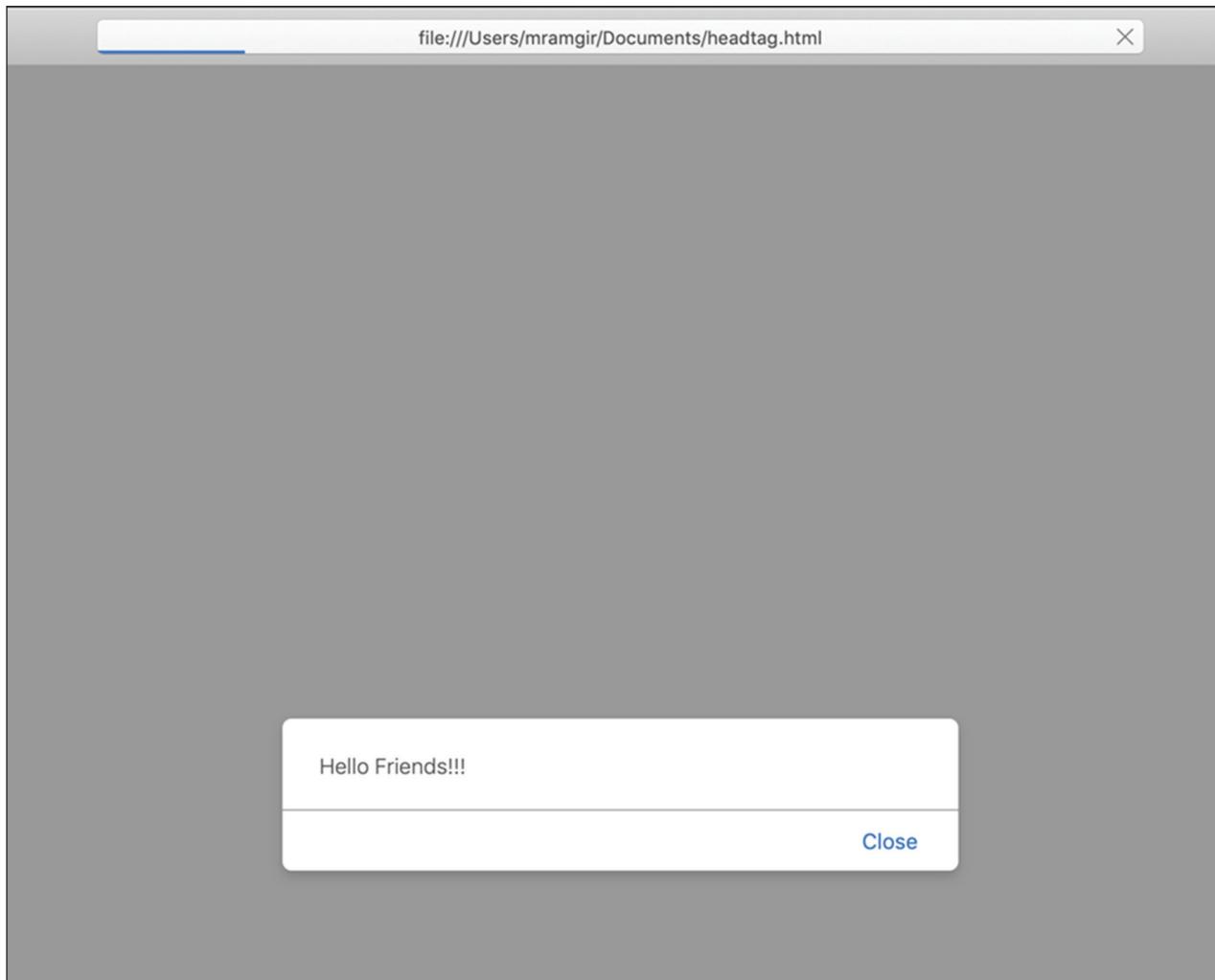
3.3.7 <head> Tag

The *<head>* element, as the name suggests, is the header section of an HTML document. This element contains other tags like *<title>*, *<meta>*, *<script>*, and *<style>* used for describing metadata information of the page such as title, description, keywords, etc. Each tag has a specific purpose. The content of these tags is not visible on the page; the only tag whose content is visible is the *<title>* tag that puts the content on the browser title bar.

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First HTML Example</title>
        <meta charset="UTF-8">
        <script>
            alert("Hello Friends!!!");
        </script>
        <style>
            h1 {
                color: red;
                text-align: center;
            }
        </style>
    </head>
    <body>
        <h1>HEAD tag example with Metadata</h1>
        <p>This meta data information is not visible on a page</p>
    </body>
</html>
```

The preceding code contains all the tags we have discussed. You can check the title window of the browser; it will show "My First HTML Example" and see the effect of *<style>* tag on the header and text. Also, it contains *<meta>* tag which sets the

charset to UTF-8. Then it has `<script>` tag which adds JavaScript alert statement. This shows an alert box as soon as you load the page in a browser window. The preceding code shows the following result in a browser window.



Once you click on the close button on the alert box, the box will disappear and the following window will appear.



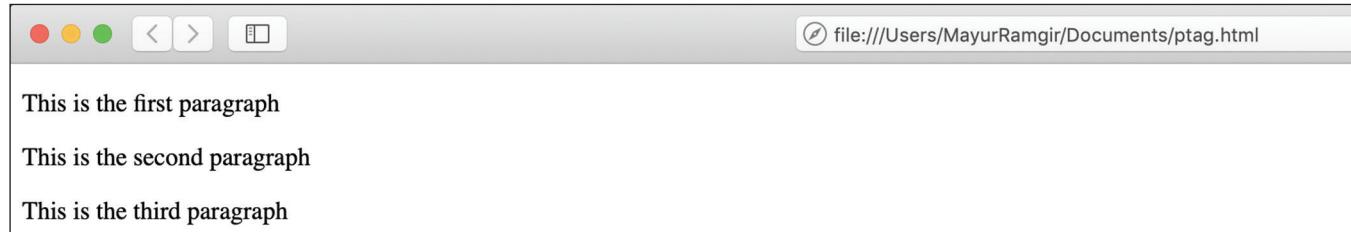
Can you add more than one `<head>` tag?

3.3.8 Paragraphs

To describe paragraphs, `<p>` is used. It is generally used for adding a space between two sections of the text. However, how much spacing is rendered is based on the browsers. Some may render space, while others may not, so it depends. See the following example to understand it better and try to run this code in different browsers to see the effect of `<p>` tag.

```
<!DOCTYPE html>
<html>
  <body>
    <p>This is the first paragraph</p>
    <p>This is the second paragraph</p>
    <p>This is the third paragraph</p>
  </body>
</html>
```

The preceding program produces the following result.

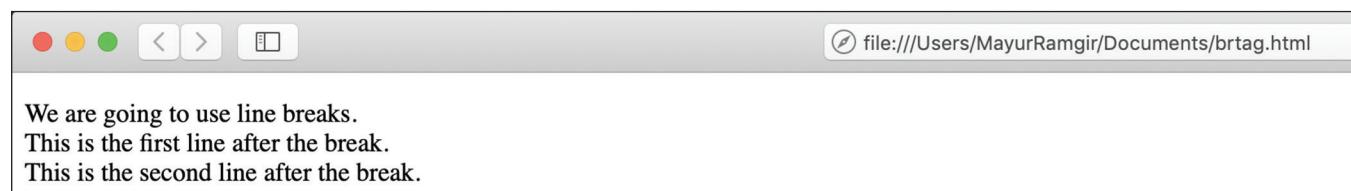


3.3.9 Line Breaks

To divide lines in HTML, `
` is used for a line break. This element is categorized as a “void element” which means it does not contain any content. Hence, it does not require any closing tag. Note that the intention of using `
` tag should be for breaking the content in which it is used. It should not be used for paragraph breaks. For example, `
` is good to use in addresses, songs, etc. Let us see the following example,

```
<!DOCTYPE html>
<html>
  <body>
    <p>
      We are going to use line breaks.<br>This is the first line after
      the break. <br>This is the second line after the break.
    </p>
  </body>
</html>
```

As you might have noticed, the use of `
` does not require a closing tag. This code produces the following result.



3.4 | Text Formatting Tags

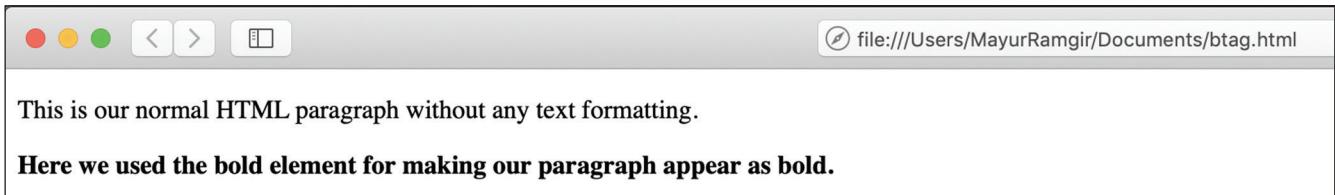
HTML provides web designers with the capability for infusing texts with different meanings. Elements like ** and *<i>* can be used for the formatting of output.

3.4.1 Bold Text

Let us begin with bold. It is possible to make text appear bold in HTML by using **. For example,

```
<!DOCTYPE html>
<html>
  <body>
    <p>This is our normal HTML paragraph without any text formatting.</p>
    <p>
      <b>Here we used the bold element for making our paragraph appear
      as bold. </b>
    </p>
  </body>
</html>
```

Above code will produce the following output. Pay attention to the text that appears darker than the previous paragraph text.

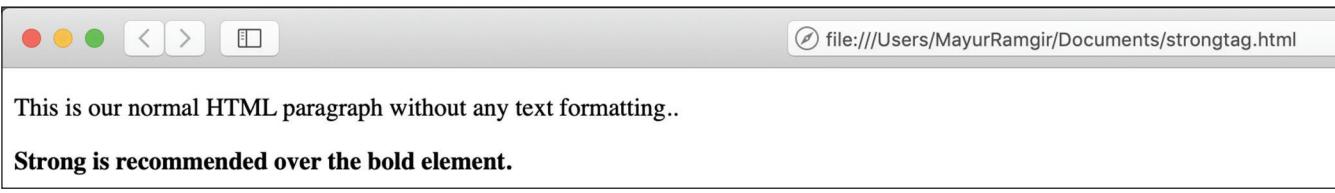


3.4.2 Strong

The ** element is similar to bold element, but it recommended over bold because it provides semantic importance to HTML elements. See the following example to understand this better.

```
<!DOCTYPE html>
<html>
  <body>
    <p>This is our normal HTML paragraph without any text formatting.</p>
    <p>
      <strong>Strong is recommended over the bold element.</strong>
    </p>
  </body>
</html>
```

The above code produces the following result, which shows that ** element has same effect as ** element.

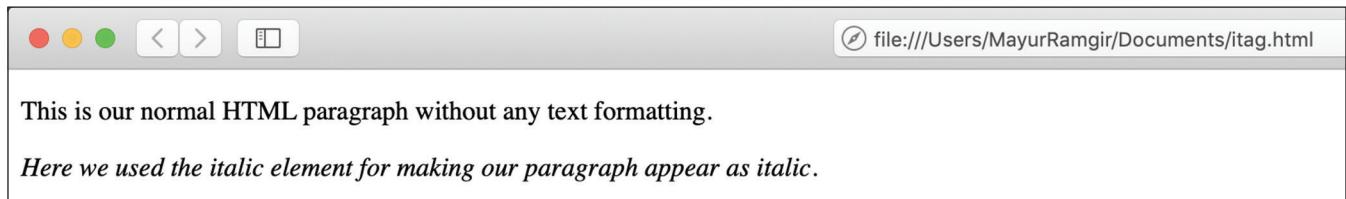


3.4.3 Italic Text

Likewise, to add italicization to your text but without any semantic importance, we use *<i>*. See the following example which shows how *<i>* tag can be used inside any other tag as well and still produce the desired result for the enclosed text.

```
<!DOCTYPE html>
<html>
<body>
    <p>This is our normal HTML paragraph without any text formatting.</p>
    <p>
        <i>Here we used the italic element for making our paragraph appear
        as italic.</i>
    </p>
</body>
</html>
```

This code produces the following result which shows the use of *<i>* tag and how it can transform the text orientation to slightly tilted.

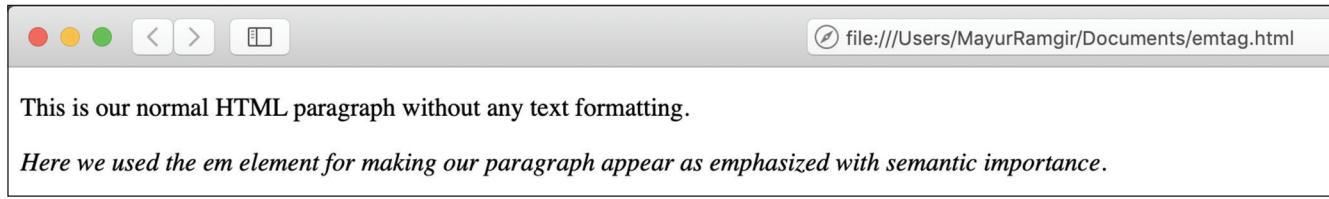


3.4.4 Emphasized Text

To add emphasis and semantic importance, we use the ** element. This element changes the text orientation and attracts attention. See the following code.

```
<!DOCTYPE html>
<html>
<body>
    <p>This is our normal HTML paragraph without any text formatting.</p>
    <p>
        <em>Here we used the em element for making our paragraph appear
        as emphasized with semantic importance.</em>
    </p>
</body>
</html>
```

Above code produces the following result. See how the text looks changed. You can also use ** tag on words or characters.

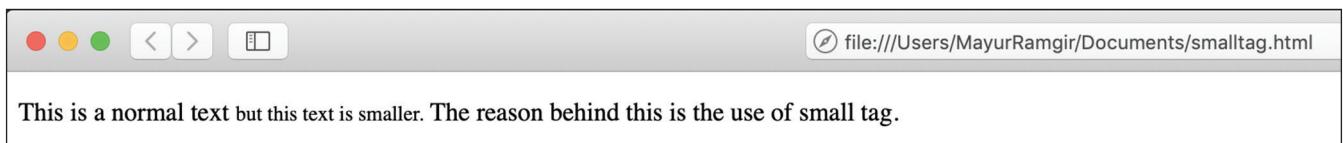


3.4.5 Small Text

To decrease the size of a word or a piece of text, `<small>` is used. This tag makes the text look smaller compared to other text. See the following example which shows the use of `<small>` tag that can be used in between any other tags.

```
<!DOCTYPE html>
<html>
  <body>
    <p>
      This is a normal text <small>but this text is smaller.</small> The
      reason behind this is the use of small tag.
    </p>
  </body>
</html>
```

This program produces the following output.



3.4.6 Marked Text

For highlighting or marking a piece of text, we use `<mark>` element. This tag highlights the text. See the following example which shows the use of `<mark>` tag.

```
<!DOCTYPE html>
<html>
  <body>
    <h2>
      This portion of the heading is unmarked
      <mark> But This is Marked</mark>
      Due to the use of mark element
    </h2>
  </body>
</html>
```

This code produces the following result which shows how easy it is to highlight a word or complete text with the help of `<mark>` tag.



3.4.7 Removed Text

To show removed or deleted text, the HTML element `` is used. For example,

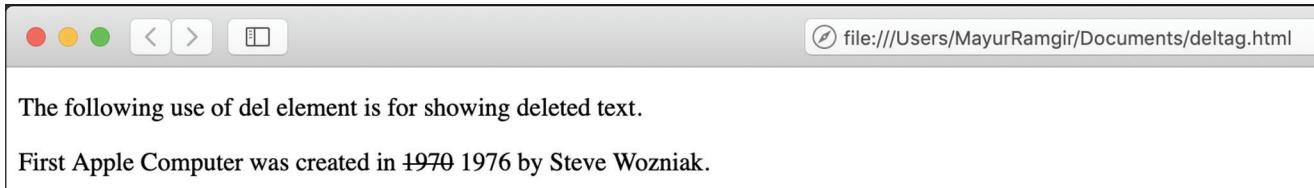
```
<!DOCTYPE html>
<html>
<body>

<p>The following use of del element is for showing deleted text.</p>

<p>
    First Apple Computer was created in
    <del>1970</del>
    1976 by Steve Wozniak.
</p>

</body>
</html>
```

This program produces the following result, which shows the use of `` tag to highlight the word as deleted.



3.4.8 Inserted Text

The `<ins>` tag is used to show the inserted text. For example,

```
<!DOCTYPE html>
<html>
<body>
    <p>The following "ins" element represents added text (inserted)</p>
    <p>
        In this paragraph this text is normal and the following text is
        <ins>inserted text</ins>
        Again we start with a normal text and now we will use the following as
        <ins>inserted text</ins>
        .
    </p>
</body>
</html>
```

This code produces the following result which shows the inserted text. Notice the text that we have used in `<ins>` tag.

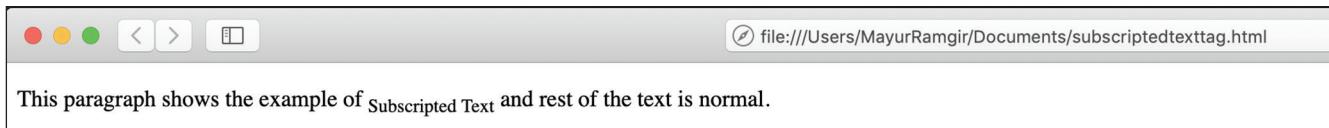


3.4.9 Subscripted Text

In order to display subscripted text, we use `<sub>` tag. This allows showing a desired text as subscripted text. See the following code to understand the use of `<sub>` better.

```
<!DOCTYPE html>
<html>
  <body>
    <p>
      This paragraph shows the example of <sub>Subscripted Text</sub> and rest of
      the text is normal.
    </p>
  </body>
</html>
```

This program produces the following result.

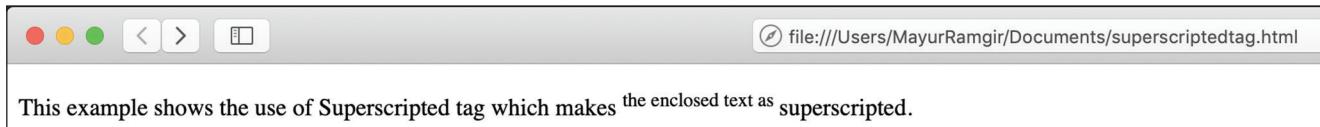


3.4.10 Superscripted Text

The `<sup>` element is similar to `<sub>` except that it displays superscripted text. Let us see the following example,

```
<!DOCTYPE html>
<html>
  <body>
    <p>
      This example shows the use of Superscripted tag which makes <sup>the enclosed
      text as </sup> superscripted.
    </p>
  </body>
</html>
```

This code produces the following result.



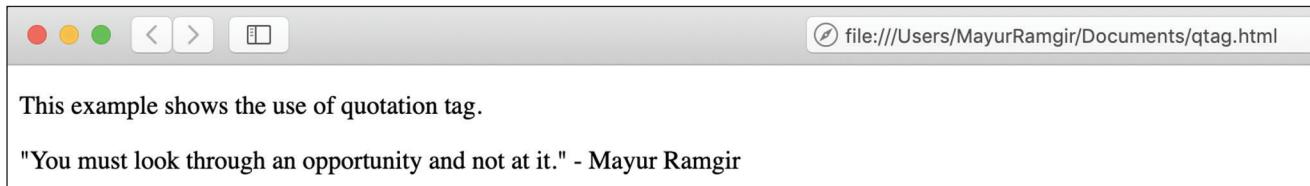
3.5 | Quotations

3.5.1 Short Quotations

To represent a quotation which is short, the `<q>` tag is used. When browsers process it, they append quotation marks accordingly. Let us see the following example,

```
<!DOCTYPE html>
<html>
<body>
    <p>This example shows the use of quotation tag.</p>
    <p>
        <q>You must look through an opportunity and not at it.</q> - Mayur Ramgir
    </p>
</body>
</html>
```

This code produces the following result which shows the use of quotation tag to add double quotes to the enclosing text.



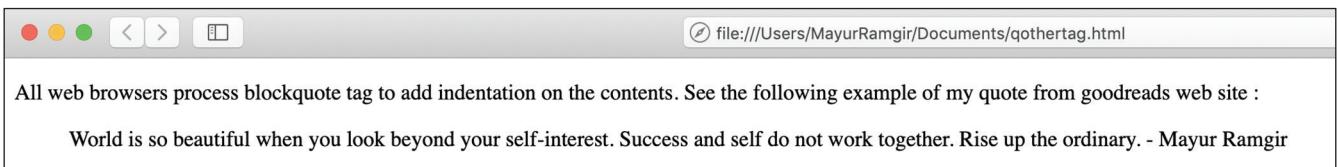
Why should you use `<q>` tag rather than just adding double quotes to the text?

3.5.2 Quotation from Other Sources

To represent a quotation from a different source, `<blockquote>` is used. When browsers find this tag, they automatically apply indentation to its contents. Let us see the following example,

```
<!DOCTYPE html>
<html>
<body>
    <p>All web browsers process blockquote tag to add indentation on
       the contents. See the following example of my quote from goodreads web site :
    </p>
    <blockquote
        cite="https://www.goodreads.com/author/quotes/15052586.Mayur_Ramgir">
        World is so beautiful when you look beyond your self-interest. Success and self do not
        work together. Rise up the ordinary. - Mayur Ramgir
    </blockquote>
</body>
</html>
```

This code produces the following result.



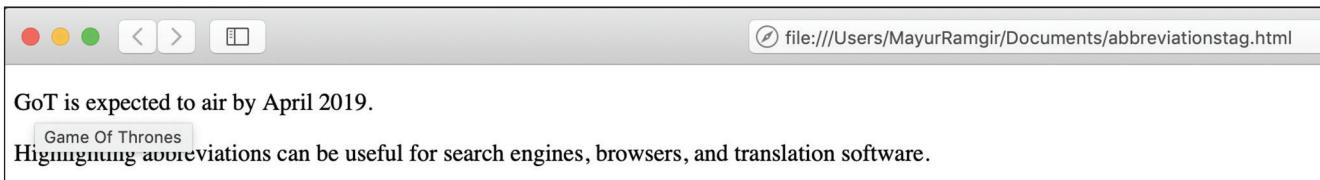
Observe how the reference of the source is attached in the tag with “cite”.

3.5.3 Abbreviations

To highlight an abbreviation, `<abbr>` is used. But why engage in such practice rather simply writing it as part of the text? Well doing so can prove to be useful for search engines, browsers, and translation software for getting any crucial bit of information. See the following example which shows the use of `<abbr>` tag.

```
<!DOCTYPE html>
<html>
  <body>
    <p>
      <abbr title="Game Of Thrones">GoT</abbr> is expected to air by April
      2019.
    </p>
    <p>Highlighting abbreviations can be useful for search engines,
       browsers, and translation software.</p>
  </body>
</html>
```

This code produces the following result which shows the use of `<abbr>`. If you hover over the abbreviated text, it will show you the content.



3.5.4 Contact Information

To get information about the author of an article or document, `<address>` is used. The contents of this element are displayed in italic text. Generally, a line break precedes and follows the element in all the major browsers. Let us see the following example,

```
<!DOCTYPE html>
<html>
  <body>
    <p>See the following example of the address tag.</p>
    <address>
      Apple, Inc.<br> Visit us at:<br> Cupertino,<br>
      California, United States<br>
    </address>
  </body>
</html>
```

This code produces the following result which shows the use of `<address>` tag to format the address.



3.6 | Comments

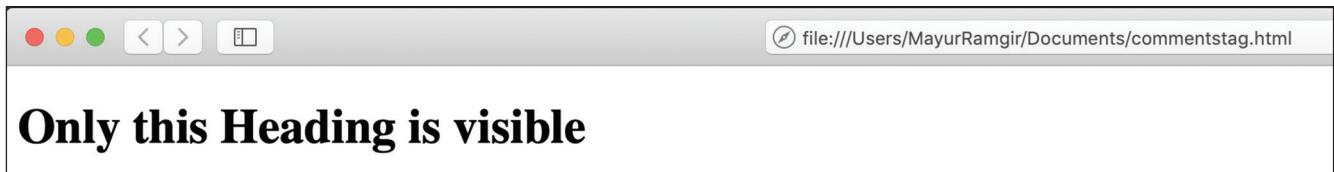
Like the syntax of programming languages, the markup-based syntax also supports comments with the help of a comment tag. To insert a comment, you can use the following format:

```
<!--The comment body -->
```

If you are unfamiliar with comments, then do note that these comments are not shown by browsers. So what is their use? Well, they allow web designers to add a reminder or describe a useful piece of information. In large code-bases, comments are a must-have component for web designers to track their use of certain programming elements while it also assists others in their team to get an understanding of the code. Let us see the following comment example,

```
<!DOCTYPE html>
<html>
  <body>
    <!-- This is an example of a comment -->
    <h1>Only this Heading is visible</h1>
    <!-- You can add as many comments as you like -->
  </body>
</html>
```

This code produces the following result. In this example, you can see that the comments are not visible to the user.



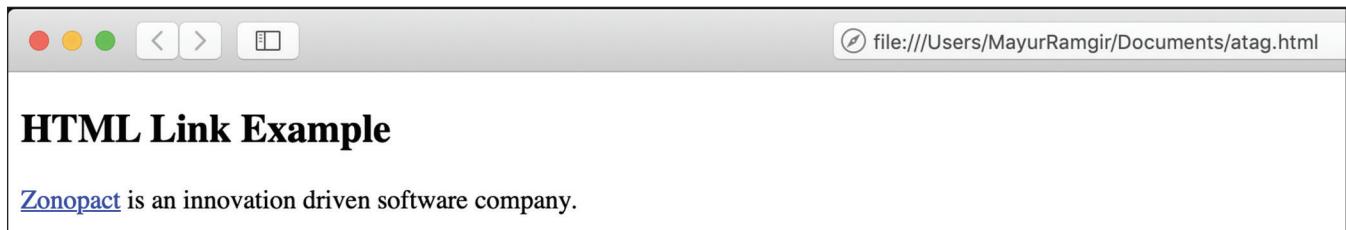
Can you show another HTML page on your page?

3.7 | Links

Have you ever wondered how the blue-colored under-lined text on websites takes you to a separate webpage? Well, the mechanism works because of HTML's links. More technically, these links are called hyperlinks which facilitate users to go from one document to another. When a mouse is hovered around such links, the cursor transforms into a mini hand. A link can contain any content like text, images, or an HTML element. These links are formed for the above mentioned type of content by specifying URLs in the "href" attribute of [tag or also called *anchor tag*. To try a simple example, write the following:](#)

```
<!DOCTYPE html>
<html>
  <body>
    <h2>HTML Link Example</h2>
    <p>
      <a href="https://www.zonopact.com">Zonopact</a> is an innovation
      driven software company.
    </p>
  </body>
</html>
```

The attribute “`href`” stands for Hypertext Reference and it requires the webpage to re-direct the user to a link. After adding a reference, we can add any content on which the hyperlink has to be placed. For external website resources, we can use the above shown link format like adding “`https://www.`” in front of the URL. However, for links of the same website in which this anchor tag is used, we can simply call a relative URL, which means we do not need to add “`https://www.`” This code produces the following result.



A screenshot of a web browser window. The title bar says "file:///Users/MayurRamgir/Documents/atag.html". The main content area contains the heading "HTML Link Example" and the text "[Zonopact](#) is an innovation driven software company."

QUICK CHALLENGE

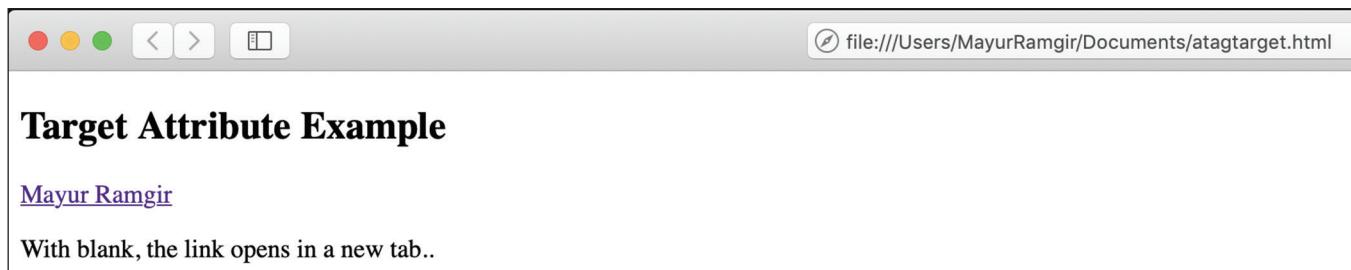
Add nested hyperlink tags and open this page in a browser. Note your observations.

3.7.1 The Target Attribute

To add more customization in links, you can use the “`target`” attribute. This attribute helps to point the browser into the tab where your link should be opened. It can have only the following values: `_blank`, `_self`, `_parent`, `_top`, and `framename`. We give an example of `_blank`; see the following code. Here, our link opens in a new tab. By default, links are set to `_self` which means that the link opens in the same tab. This attribute is also used in other HTML elements like `form` which we would cover later in our examples.

```
<!DOCTYPE html>
<html>
  <body>
    <h2>Target Attribute Example</h2>
    <a href="https://www.mayurramgir.com" target="_blank">Mayur Ramgir</a>
    <p>With blank, the link opens in a new tab..</p>
  </body>
</html>
```

This code produces the following result.



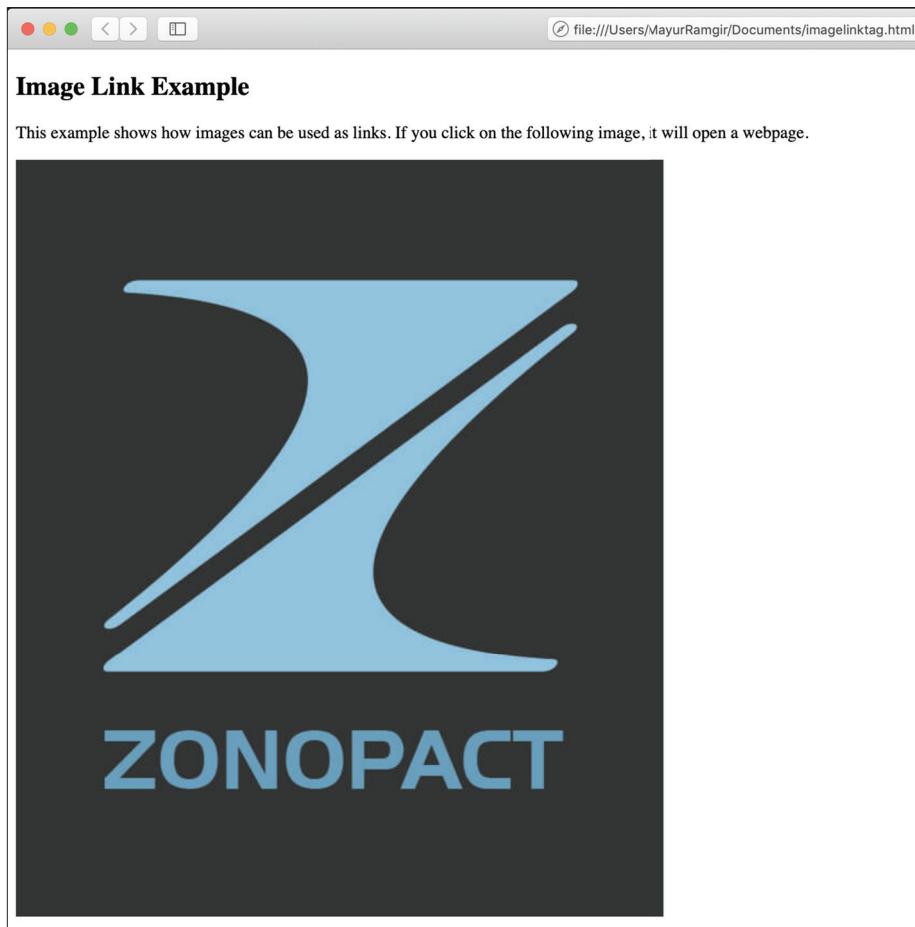
A screenshot of a web browser window. The title bar says "file:///Users/MayurRamgir/Documents/atagtarget.html". The main content area contains the heading "Target Attribute Example", a link "[Mayur Ramgir](#)", and the text "With blank, the link opens in a new tab..".

3.7.2 Images as Links

So far we have only used text as hyperlinks. Now, we will go over an example of an image link which means an image which can open another link upon clicking. You can simply refer to them as “clickable images”.

```
<!DOCTYPE html>
<html>
<body>
    <h2>Image Link Example</h2>
    <p>This example shows how images can be used as links. If you click
        on the following image, it will open a webpage.</p>
    <a href="https://www.zonopact.com"> 
    </a>
</body>
</html>
```

The preceding code produces the following result which shows how elements can be included in `<a>` tag.



3.7.3 Bookmark

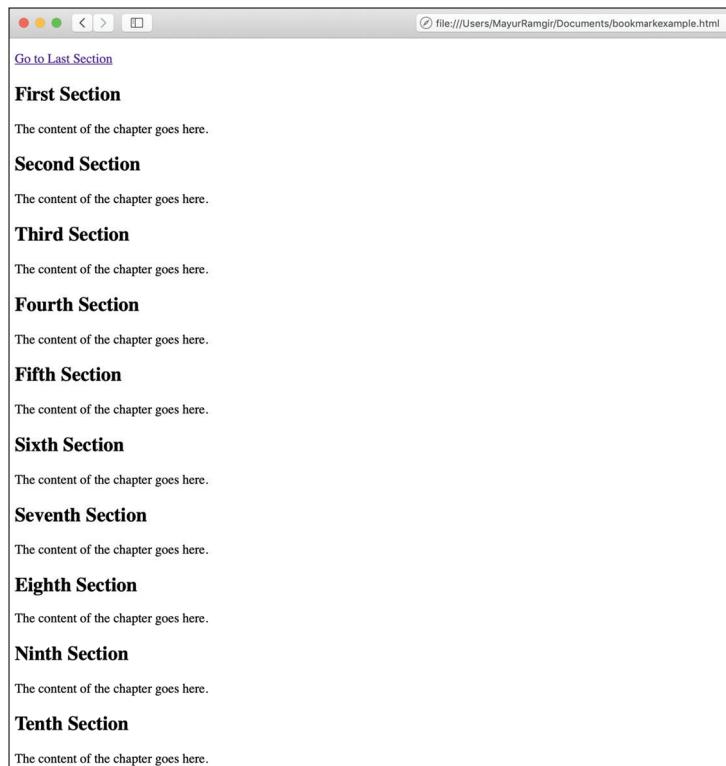
For longer pages where the user has to scroll a lot to get the desired information, bookmark comes as a handy tool. It allows developers to create sections where the user can directly go to by clicking on a link which looks like an on-page menu. To create a bookmark, you will first require to add the id attribute to the element like `<div id="myId"></div>`. Then you can create an anchor tag with desired content text like “Go to My Section” and add the id of the element in the href attribute like `Go to My Section`. See the following example to understand this better.

```

<!DOCTYPE html>
<html>
  <body>
    <p>
      <a href="#10">Go to Last Section</a>
    </p>
    <h2>First Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2>Second Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2>Third Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2 id="4">Fourth Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2>Fifth Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2>Sixth Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2>Seventh Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2>Eighth Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2>Ninth Section</h2>
    <p>The content of the chapter goes here.</p>
    <h2 id="10">Tenth Section</h2>
    <p>The content of the chapter goes here.</p>
  </body>
</html>

```

See how we used a link to directly go over our last section.



3.8 | Images

In the example presented in Section 3.7.2, we used an image. Now let us explore this in detail. Images can be placed in HTML code to improve the appearance of web pages. Usually, only text content can be pretty bland and boring for visitors. Hence, images are added to tempt them into spending more time on websites. Images require `` tag with a “src” or source for the source location (website address) of the image. We can then use the properties of “height” and “width” to re-size our image according to our requirements. Let us check how we placed pretty Grand Canyon on our website.

```
<!DOCTYPE html>
<!DOCTYPE html>
<html>
<body>
    <h1>Image Example</h1>
    
</body>
</html>
```

This code shows the following result in a browser window.

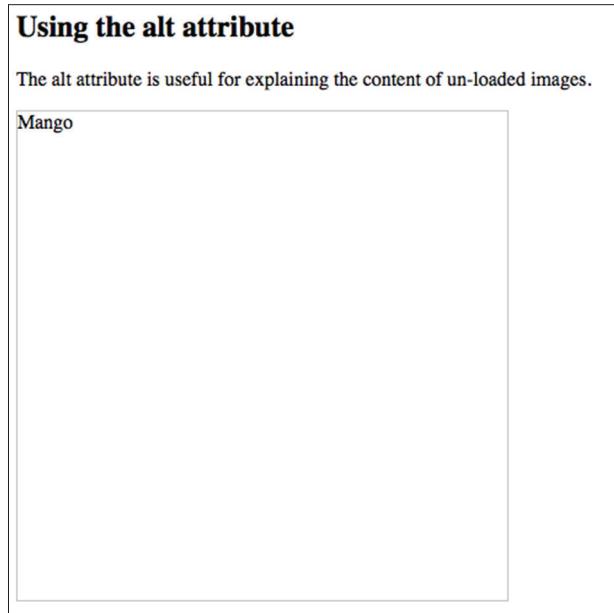


3.8.1 Alt Attribute

In some cases, an image cannot be loaded in the web browser of the user because of various reasons such as Internet connection problem or the wrong insertion of the image source address. In order to counter such a situation, the “alt” attribute is used. It is simply a description of the image to inform the users what is missing. For instance

```
<!DOCTYPE html>
<html>
<body>
    <h2>Using the alt attribute</h2>
    <p>The alt attribute is useful for explaining the content of un-loaded images.</p>
    
</body>
</html>
```

This code shows the following result in a browser window.



3.9 | Tables

HTML also supports the inclusion of tables with the `<table>` tag. The `<tr>` tag specifies the table row while the `<th>` tag specifies the table header. Usually, table headings are centered and bold. The cell or data in the tag is specified by using the `<td>` tag which can take text, lists, images, and even other tables. For example,

```
<!DOCTYPE html>
<html>
<body>
    <h1>HTML Table</h1>
    <table style="width: 70%">
        <tr>
            <th align="left">Employee Name</th>
            <th align="left">Designation</th>
            <th align="left">City</th>
        </tr>
        <tr>
            <td>Jack</td>
            <td>Salesperson</td>
            <td>Austin</td>
        </tr>
        <tr>
            <td>Chad</td>
            <td>Accountant</td>
            <td>Fall River</td>
        </tr>
        <tr>
            <td>Daniel</td>
            <td>Software Developer</td>
            <td>Fairfax</td>
        </tr>
    </table>
</body>
</html>
```

This code shows the following result in a browser window.

HTML Table		
Employee Name	Designation	City
Jack	Salesperson	Austin
Chad	Accountant	Fall River
Daniel	Software Developer	Fairfax

Now, we do have the basic table but it can be improved. Add a border property with the `<style>` tag. Let us change our above example as follows:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    table, th, td {
      border: 1px solid red;
    }
  </style>
</head>
<body>
  <h1>HTML Table</h1>
  <table style="width:70%">
    <tr>
      <th align = "left">Employee Name</th>
      <th align = "left">Designation</th>
      <th align = "left">City</th>
    </tr>
    <tr>
      <td>Jack</td>
      <td>Salesperson</td>
      <td>Austin</td>
    </tr>
    <tr>
      <td>Chad</td>
      <td>Accountant</td>
      <td>Fall River</td>
    </tr>
    <tr>
      <td>Daniel</td>
      <td>Software Developer</td>
      <td>Fairfax</td>
    </tr>
  </table>
</body>
</html>
```

This code shows the following result in a browser window.

HTML Table		
Employee Name	Designation	City
Jack	Salesperson	Austin
Chad	Accountant	Fall River
Daniel	Software Developer	Fairfax

3.10 | Lists

In HTML, we have two types of lists:

1. Unordered list which means items are arranged in bullet form.
2. Ordered list which means items are arranged in a numeric or orderly form.

To make an unordered list, you simply need to use the `` tag. For the items of the tag, you can use `` to specify each individual item. For example,

```
<!DOCTYPE html>
<html>
<body>
    <h1>What do you want to eat for dinner?</h1>
    <ul>
        <li>Pizza</li>
        <li>Burger</li>
        <li>Sandwich</li>
    </ul>
</body>
</html>
```

The preceding code shows the following result in a browser window.

What do you want to eat for dinner?

- Pizza
- Burger
- Sandwich

HTML's unordered list provides customization which means that you can also try other types of bullets for your lists. For instance, you can use `list-style-type` to add the following types of bullets: disc, circle, square, and none. An example with "square" style is as follows:

```
<!DOCTYPE html>
<html>
<body>
    <h1>What do you want to eat for dinner?</h1>
    <ul style="list-style-type: square;">
        <li>Pizza</li>
        <li>Burger</li>
        <li>Sandwich</li>
    </ul>
</body>
</html>
```

The above code shows the following result in a browser window.

What do you want to eat for dinner?

- Pizza
- Burger
- Sandwich

On the other hand, we have an ordered list which requires the use of `` tag. For example,

```
<!DOCTYPE html>
<html>
<body>
    <h1>What do you want to eat for dinner?</h1>
    <ol>
        <li>Pizza</li>
        <li>Burger</li>
        <li>Sandwich</li>
    </ol>
</body>
</html>
```

This code shows the following result in a browser window.

What do you want to eat for dinner?

1. Pizza
2. Burger
3. Sandwich

Ordered lists can be customized too. You can add the following “type” properties: 1, A, a, I, and i. To experiment with a practical demonstration, try this code.

```
<!DOCTYPE html>
<html>
<body>
    <h1>What do you want to eat for dinner?</h1>
    <ol type="A">
        <li>Pizza</li>
        <li>Burger</li>
        <li>Sandwich</li>
    </ol>
</body>
</html>
```

The above code shows the following result in a browser window.

What do you want to eat for dinner?

- A. Pizza
- B. Burger
- C. Sandwich

HTML also has a list type called *description list*. These types of lists have multiple keywords. The `<dl>` tag specifies the list, `<dt>` tag specifies the name of the item, and `<dd>` tag is used to write down the description of the items.

```
<!DOCTYPE html>
<html>
<body>
    <h1>A Description List</h1>
    <dl>
        <dt>AI</dt>
        <dd>- A branch of computer science which mainly deals with making computers "intelligent" or human-like.</dd>
        <dt>Internet of Things</dt>
        <dd>- One of the latest IT technologies which aims at revolutionizing the world by embedding internet and computer hardware in all types of objects.</dd>
    </dl>
</body>
</html>
```

This code shows the following result in a browser window.

A Description List

AI

- A branch of computer science which mainly deals with making computers "intelligent" or human-like.
- Internet of Things
 - One of the latest IT technologies which aims at revolutionizing the world by embedding internet and computer hardware in all types of objects.



Which markup language does OpenSearch, RSS and Atom use?



3.11 | Attributes to Style HTML Elements

In this section, we will discuss the attributes that are used to style elements like changing color, adding font size, etc. These are CSS properties that you can add to HTML elements directly. Though we will study CSS in detail in the Chapter 4, here you will introduce a few things about CSS styling and adding directly in the HTML code.



What are Microformats and how do you define Microformats patterns?

3.11.1 Style Attribute

To add some colors and styling in an HTML element, we use an attribute called style. Such an attribute works by using the following format:

```
<tagname style="property:value;">
```

For instance, to change the color of background to green, we can do the following:

```
<!DOCTYPE html>
<html>
<body style="background-color: green;">
    <h1>Using the property background-color!</h1>
    <p>Changing the color of the background from white to green.</p>
</body>
</html>
```

This code shows the following result in a browser window, which has a green background. Since this book is printed in black and white, the color is not visible. However, if you run this code in a browser window, you will be able to see the color.

Using the property background-color!

Changing the color of the background from white o green.

3.11.2 Text Color

To change the color of the text to red for an HTML element, we can write the following:

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="color: red;">Red Heading</h1>
    <p style="color: red;">By using style attribute and its property
color, we have changed the color of our paragraph text to red.</p>
</body>
</html>
```

The above code shows the following result in a browser window, which has text in color red. Since this book is printed in black and white, the color is not visible. However, if you run this code in a browser window, you will be able to see the color.

Red Heading

By using style attribute and its property color, we have changed the color of our paragraph text to red.

3.11.3 Fonts

Similarly, style can also be used to add fonts for our text by using the property “font-family”.

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="font-family: courier;">Heading is in verdana.</h1>
    <p style="font-family: verdana;">Paragraph is in courier..</p>
</body>
</html>
```

The preceding code shows the following result in a browser window.

Heading is in verdana.

Paragraph is in courier..

3.11.4 Size of Text

We can also use `<style>` for increasing the size of our font. For instance,

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="font-size: 200%;">Increasing our font by 200 percent</h1>
    <p style="font-size: 100%;">Increasing our font by 100 percent.</p>
</body>
</html>
```

The preceding code shows the following result in a browser window.

Increasing our font by 200 percent

Increasing our font by 100 percent.

3.11.5 Text Alignment

To align our elements to the left, right, or in the center, we use `text-align` property of `<style>`. For example,

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="text-align: left;">This heading is aligned to the left.</h1>
    <h2 style="text-align: right;">This heading is aligned to the right.</h2>
    <h3 style="text-align: center;">This heading is aligned to the center</h3>
</body>
</html>
```

This code shows the following result in a browser window.

This heading is aligned to the left.

This heading is aligned to the right.

This heading is aligned to the center

3.11.6 Different Screen Sizes

It is hard to ascertain how exactly HTML would look over different screens or when the window is resized. In HTML, extra lines or spaces cannot be added for modifying the output because the browser eliminates any additional lines and spaces. For example

```
<!DOCTYPE html>
<html>
<body>
    <p>Here we used many lines but the browser does not process it so  

this means that the browser does not take lines into account.</p>
    <p>Here in this paragraph we used too many spaces but the, browser  

did not process it.</p>
</body>
</html>
```

This code shows the following result in a browser window.

Here we used many lines but the browser does not process it so this means that the browser does not take lines into account.
Here in this paragraph we used too many spaces but the, browser did not process it.

3.11.7 No Closing Tag

Most of the times, a tag is processed correctly even without adding its closing tag, just like in the following example,

```
<!DOCTYPE html>
<html>
<body>
    <p>Paragraph without the closing tag.
    <p>Paragraph without the closing tag.
    <p>Paragraph without the closing tag.
    <p>But still always use the closing tag</p>
</body>
</html>
```

This code shows the following result in a browser window.

Paragraph without the closing tag.
Paragraph without the closing tag.
Paragraph without the closing tag.
But still always use the closing tag

However, still it is recommended that you always make use of the closing tag because sometimes unexpected errors can be caused without it.

Summary

This chapter is more focused on understanding one of the most dominant front-end technologies known as HTML. HTML stands for HyperText Markup Language and is the only widely accepted markup language for website creation. It offers various helper tags that are interpreted by all the web browsers to render a webpage. There are no competitors to HTML but there are a few alternatives like Flash, Flex, Java Swing, Silverlight, etc. However, these are not as widely adopted by all the platforms as HTML. Many times they need a special plugin to install on the client browser to run a webpage successfully.

In this chapter, we have learnt the following concepts:

1. HTML and the basic building blocks of HTML.
2. Elements, tags, and attributes.
3. Usage of various tags like table, label, p, a, etc. to design a webpage.
4. How to use basic CSS styles in HTML code itself which is known as inline styling practice.
5. Styling text and setting color, size, etc.

In the next chapter, we will learn about CSS (Cascading Style Sheets) that is used for styling HTML elements on webpages. CSS is used to style the HTML elements and make them look nice. CSS is also helpful in arranging data in a presentable manner. We will explore various CSS elements and their uses.

Multiple-Choice Questions

1. The accesskey shows a specific keyboard navigation for the element?
 - (a) True
 - (b) False
2. Which one of the following helps us to define a visible heading for the details element?
 - (a) <brief>
 - (b) <main>
 - (c) <mark>
 - (d) <summary>
3. The _____ element is utilized to render modest graphics like a line art, graphs, and other custom graphical units on the client side.
 - (a) CSS
4. In HTML Video/Audio DOM, _____ returns or sets the CORS settings of the video/audio.
 - (a) duration
 - (b) currentTime
 - (c) defaultPlaybackRate
 - (d) crossOrigin
5. HTML code consists of _____.
 - (a) Attributes
 - (b) Elements
 - (c) Tags
 - (d) All of the above

Review Questions

1. How many heading levels does HTML offer?
2. How many types of targets can you set on a hyperlink <a> tag?
3. Which element in HTML5 performs line break?
4. Can HTML element have no closing tag?
5. How can one change a font size of a particular element?
6. Which attribute can you use to set CSS style on an element?
7. How can you make an image clickable to open a new page in a new browser tab?
8. How can you show a text in italic mode?
9. Which metadata makes a page to zoom to twice its natural size by allowing to set a value of initial-scale=2?
10. Which element offers a provision to express and annotate East Asian character pronunciation?
11. Why should you use instead of tag?
12. What is the difference between <script> and <style> element?
13. Can you use <style> element inside <body> tag?
14. Can you have nested hyperlink tags <a>? For example, Mayur Ramgir from Zonopact
15. Can search engine index your page if you miss <meta> tag on your page?

Exercises

1. Create a splash screen by adding an image and giving it a hyperlink which will open a new page in the same browser window.
2. Create a page that shows famous quotes from famous people. Make sure the quotes are displayed properly in format which looks like a quote oppose to just a plain text.
3. Create a page which has 5 hyperlinks and each hyperlink opens a new page in the same browser. That new page has a link titled as “Go Back” that takes you back to this main page.

Project Idea

Use HTML5 elements to create a dictionary of 20 most unused words. This dictionary website will have a home page which shows all 20 words with hyperlinks to open each word in a new page. This new page will have a link that will take user back to

the home page. On the individual words' pages, use list element to show the uses of that words, use table to show synonyms and antonyms, use various font styles like bold, italic, underline, etc., and highlight a word if used in a description.

Recommended Readings

1. Thomas Powell. 2017. *HTML & CSS: The Complete Reference, Fifth Edition*. McGraw Hill Education: New York
2. Mike McGrath. 2017. *HTML5 In Easy Steps*. BPB Publications: New Delhi
3. Julie C. Meloni. 2018. *HTML, CSS, and JavaScript All in One: Covering HTML5, CSS3, and ES6, Sams Teach Yourself, 3rd Edition*. Sams Publishing: Indiana
4. The World Wide Web Consortium (W3C) – <https://www.w3.org/html/>
5. W3School – <https://www.w3schools.com/html/>

Introduction to Cascading Style Sheets

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- CSS and how to use it.
- How CSS can transform the look of a page.
- CSS configuration with HTML.
- Advantages of CSS.
- Various selectors like Id, class, etc. and how to use them.
- How to attach an external CSS file.
- Various HTML elements styles.

4.1 | Introduction

In Chapter 3, we have gained a firm grasp of HTML. Now, let us proceed to the next fundamental front-end technology – Cascading Style Sheets (CSS).

CSS falls into the domain of style sheet languages. It is primarily used for defining the presentation of web pages. While HTML defines the structure of the complete web page, CSS adds some style to it. Without CSS, websites look ugly and bland. Investing in CSS is a good way to add appeal to the website and increase web traffic.

We can use single CSS file with elements such as the layout, fonts, color, and other related elements of a webpage which can be written once and used many times. By using a single CSS file, the layout, fonts, color, and other related elements of a webpage can be sorted out all at once. CSS files are saved with an extension of “.CSS”.

4.1.1 History

On October 10, 1994, Håkon Wium Lie proposed CSS. During that time, Lie used to work at CERN with Tim Berners-Lee. Many other style sheet languages were also proposed at that time and the World Wide Web Consortium held discussions on the matter. After some time, CSS1 was released in 1996. This release had substantial input from Bert Bos who was the co-author and is known as the CSS' co-creator.

4.2 | Overview of CSS

As explained, CSS is a language which is used to define how users can view documents and understand how they are laid out or styled. This document is generally a text file which uses a markup language. Most of the time, this markup language is HTML. However, you may also have to use CSS with XML, Scalable Vector Graphics (SVG), or other markup languages.

By “presenting” this document we mean that it must be converted into a form which is usable by the target audience. Chrome, Firefox, and other web browsers are created with the functionality to present these documents on a computer screen, printer, or a project.

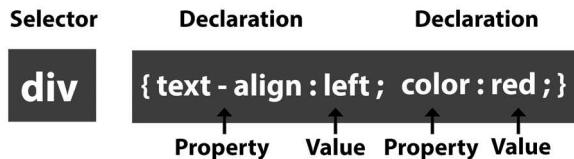
4.3 | Relationship Between HTML and CSS

The CSS rules are used by the web browsers with a document to change how it appears to the user. Such rules are designed from the following stages:

1. **Properties:** These properties are configured with values to modify how the content of HTML should be displayed to the user. For example, you can set the width of a button on the page or change its color by updating the appropriate property.

- 2. Selector:** It chooses the elements by their ids, names, types, attributes, etc., on which the desired style is going to be applied. For example, you can use it to ensure that all the paragraphs have green text color.

CSS rules in the style sheet ultimately form the look of the page.



4.4 | How Does CSS Work?

After a document is displayed by the browser, it has to use the content of the document and incorporate it with the style sheet information. The browser performs this processing using the following strategy:

1. The browser changes CSS and HTML into the Document Object Model (DOM). This model is used by computer's memory for representation of the document.
2. The DOM's contents are displayed by the browser.

The DOM model consists of a structure which resembles a tree data structure. All the attributes, elements, and text of the markup language are added as the DOM node in this structure. Each node has a specified relationship with the other. This means that a node can have a parent, child, and siblings.

If you can understand the DOM model, it can assist you to design, maintain, and debug the CSS as the DOM describes how the content of the document and the CSS have to be linked.

4.4.1 Advantages of Using CSS

1. CSS helps in saving time. It is only needed to be defined once, after which it can be used again and again to style all HTML elements for several HTML pages.
2. It helps to load web pages quicker. When CSS is used, there is no need for HTML tag attributes. Developers have to use the CSS rule for a tag one time, after which it is automatically implemented for all of the tag's occurrences. Hence, it decreases code and ultimately helps to increase the speed of downloading.
3. For a global change, you can use the CSS to just change the code in a single place which can apply the changes to all the relevant webpages. Hence, it is extremely easy to maintain CSS.
4. In comparison to HTML, CSS has a large variety of attributes which helps to add several new features and functionalities that were otherwise not possible with the use of HTML attributes.
5. CSS can help the content with optimization which means that it can display appropriately in multiple devices. CSS can utilize the same HTML which can show multiple website versions for smartphones, tablets, printers, and PDAs.
6. Styling webpages with HTML attributes is considered obsolete today as there are several disadvantages. First is the readability of the code where developers cannot easily figure out the styling of the element; adding more styling properties will become cumbersome. Second, it adds to the page load time since in case of separate CSS files, the browser can download HTML and CSS files simultaneously to save loading time. Hence, using CSS is a good design practice.

4.4.2 HTML and Styling

HTML also has several options to add "styling" on the web page, so why should developers move towards CSS? Well, in four words we can explain the practice as per our personal experience: It is a nightmare! Adding style to large websites using HTML is extremely difficult because web designers have to make modifications to each page. To tackle this issue, the World Wide Web Consortium (W3C) came up with CSS. Unlike other formatting tools, CSS has completely eliminated the need for HTML as an alternate webpage styling option.

4.5 | Syntax

With enough understanding about the necessity of CSS, let us now move to the more technical aspects of CSS.

In CSS, we have two components: a selector and a declaration block. A selector refers to any HTML element which requires styling. However, it is not necessary for a selector to be the name of an element. We can also use the attribute *id*, or *class*, as selectors.

By a declaration block, we mean the set of CSS properties with their values. These properties are separated by a colon. A declaration block may contain a single property or multiple properties. The block is terminated with a curly bracket, while the entailed properties are ended with curly braces.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p {
            color: blue;
            text-align: center;
        }
    </style>
</head>
<body>
    <p>There is a strong demand for front-end developers in the Bay Area.</p>
    <p>The paragraphs in this example are styled with the help of CSS.</p>
</body>
</html>
```

This code shows the following result in a browser window.

There is a strong demand for front-end developers in the Bay Area.

The paragraphs in this example are styled with the help of CSS.

4.5.1 The “*id*” Selector

CSS can use selectors to select a specific element to apply the desired style. For example, for an HTML element with an “*id*” attribute set to “myID”, CSS can use id selector such as #myID { color: red; } to apply the defined styling to that HTML element. In this case, the HTML element with id “myID” will get red color as defined in the CSS selector. Note that the attribute in HTML must be unique, that is, no other element should have the same id. Moreover, an id name must never begin with a digit. To use an id selector, you have to type “#” (hash character), after which you can define an element’s id. For example,

```
<!DOCTYPE html>
<html>
<head>
    <style>
        #firstpara {
            text-align: right;
            color: green;
        }
    </style>
</head>
<body>
    <p id="firstpara">Napoleon Bonaparte is widely considered to be one of the greatest
        military leaders of all time. The charismatic French emperor thwarted several
        enemies, all at once!</p>
    <p>The paragraphs in this example are styled with the help of CSS.</p>
</body>
</html>
```

This code shows the following result in a browser window.

Napoleon Bonaparte is widely considered to be one of the greatest military leaders of all time. The charismatic French emperor thwarted several enemies, all at once!
The paragraphs in this example are styled with the help of CSS.

QUICK CHALLENGE

Add multiple elements with same id and see what happens. For example, create three `<div>` tags and give them same id like `id= "myDiv"`. Then use selector to style those div elements. Note down the outcome and write lesson learned in a few sentences.

4.5.2 The “`class`” Selector

As its name suggests, this type of selector chooses elements that carry a defined class attribute. The class name is preceded by a “.”. For example,

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .styling {
            text-align: right;
            color: blue;
        }
    </style>
</head>
<body>
    <h1 class="styling">The class selector has moved the heading to the right and changed
        its color to blue.</h1>
    <p class="styling">The class selector has moved the paragraph to the right and
        changed its color to blue.</p>
</body>
</html>
```

This code shows the following result in a browser window.

The class selector has moved the heading to the right and changed its color to blue.

The class selector has moved the paragraph to the right and changed its color to blue.

There is also the flexibility to only apply changes in specific HTML elements of the class. This can be done by using the name of the element. Continuing our above example we have,

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p.styling {
            text-align: right;
            color: blue;
        }
    </style>
</head>
<body>
    <h1 class="styling">No changes were applied to the heading.</h1>
    <p class="styling">The class selector has moved the paragraph to the right and
        changed its color to blue.</p>
</body>
</html>
```

The above code shows the following result in a browser window.

No changes were applied to the heading.

The class selector has moved the paragraph to the right and changed its color to blue.

4.5.3 Groups of Selectors

Consider the following CSS code. As you can observe, all of these elements like h1, h2 and p have the same functionality, but their declaration blocks are repeated. Hence, it seems quite an un-optimized way of coding.

```
h1 {
    text-align: left;
    color: blue;
}
h2 {
    text-align: left;
    color: blue;
}
p {
    text-align: left;
    color: blue;
}
```

However, in CSS, it is possible to group selectors and incorporate similar functionality where a comma demarcates them. Therefore, we can optimize the above code and reduce our lines of CSS considerably by writing the following code:

```
<!DOCTYPE html>
<html>
<head>
    <style>
        h1, h2, p {
            text-align: left;
            color: blue;
        }
    </style>
</head>
<body>
    <h1>First Heading</h1>
    <h2>Second Heading</h2>
    <p>This paragraph is blue.</p>
</body>
</html>
```

This code shows the following result in a browser window.



4.5.4 Comments

Web designers and developers use comments in CSS to write useful details for lines of code. These notes come in handy when you want to change the code later on. Website browsers do not display comments. Comments begin with /* and end with */. For example,

```

<head>
    <style>
        p {
            color: blue;
            /* We have applied this CSS setting because of client requirements */
            text-align: left;
        }
        /* Comments
        can extend
        multiple lines */
    </style>
</head>
<body>
    <p>Comments Example</p>
    <p>Blue paragraphs.</p>
    <p>This source file has comments but the website browser would not
       show it.</p>
</body>
</html>

```

This code shows the following result in a browser window.

Comments Example
Blue paragraphs.
This source file has comments but the website browser would not show it.



Will a page throw error if CSS code is not in proper format?

4.6 | Different Methods to Integrate CSS with HTML

In this section we will discuss the various methods using which a style sheet can be inserted.

4.6.1 External Style Sheet

External style sheets are separate files that carry the **.css** extension. They simplify the presentation of webpages because they store all modifications in a single file. Style sheets are referenced in all the pages of the websites in the **<link>** element, which itself must be entailed in the **<head>** element. To make an external style sheet, open a file in Notepad and save it with “**cssexample.css**”. Add the following instructions in the file:

```

p {
    text-align: right;
    color: blue;
}

```

Now for HTML, write this code.

```

<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="cssexample.css">
</head>
<body>
    <h1>External Style Sheet Example</h1>
    <p>This paragraph is styled with CSS.</p>
</body>
</html>

```

This code shows the following result in a browser window.

External Style Sheet Example

This paragraph is styled with CSS.

Note that the CSS files do not use any of the HTML's tags.

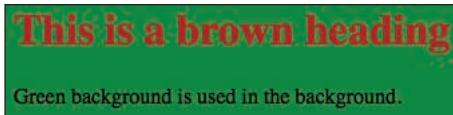
4.6.2 Internal Style Sheet

Internal style sheets are used with the HTML element `<style>` inside the `<head>` element. Remember, the `<head>` element is used for the metadata of HTML. So, why do we use internal style sheets? The purpose behind the use of internal style sheet is to add a separate or unique style to a page. For instance,

```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            background-color: green;
        }

        h1 {
            color: brown;
            margin-right: 100px;
        }
    </style>
</head>
<body>
    <h1>This is a brown heading</h1>
    <p>Green background is used in the background.</p>
</body>
</html>
```

The above code shows the following result in a browser window, where the background is shown in green and the main heading text is in red.



4.6.3 Inline Style

The third method to use CSS is inline styling. In this case, styling is applied on single elements for unique functionality. They are incorporated in HTML elements by adding the `style` attribute in the desired element. This `style` attribute can now make use of any CSS property. For instance,

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="color: green; margin-right: 30px;">This is a green heading with inline
    styles.</h1>
    <p>No CSS is used here.</p>
</body>
</html>
```

The above code shows the following result in a browser window.

This is a green heading with inline styles.

No CSS is used here.

4.6.3.1 Use of Multiple Style Sheets

Sometimes, different style sheets refer to the same HTML element. A style sheet may force a heading to go blue while another one to turn green. So how would the browser know which one to choose? In such cases, the last read style sheet is taken into account for the changes.

4.7 | Colors

In CSS, colors are defined in multiple ways. You can use common color names, or you can add RGBA, HSLA, HSL, HEX, and RGB color values.

4.7.1 Color Names

All elements in HTML can have their own background color by using the “*background-color*” property. For instance,

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="background-color: LightGray;">History of Computer
        Science</h1>
    <p style="background-color: Violet;">Among the early inventors of computers, Lady Ada
        Lovelace was perhaps one of the most remarkable one. She is often credited to be the
        first ever programmer. Additionally, she was the first one to use an algorithm.</p>
</body>
</html>
```

This code shows the following result in a browser window.

History of Computer Science

Among the early inventors of computers, Lady Ada Lovelace was perhaps one of the most remarkable one. She is often credited to be the first ever programmer. Additionally, she was the first one to use an algorithm.

Similarly, the text of the HTML elements can also be updated by using the “*color*” property.

```
<!DOCTYPE html>
<html>
<body>
    <h3 style="color: orange;">History of Computers</h3>
    <p style="color: slateblue;">Charles Babbage invented the first mechanical computer.
        Perhaps, this is why he got the title of "Father of Computers". Modern computers
        are based on his analytical machine.</p>
    <p style="color: dodgerblue;">Among the early inventors of computers, Lady Ada
        Lovelace was perhaps one of the most remarkable one. She is often credited
        to be the first ever programmer. Additionally, she was the first one to use an
        algorithm.</p>
</body>
</html>
```

This code shows the following result in a browser window where h3 text is shown in orange, first p element text is shown in slate blue and second p element text is shown in Dodger blue.

History of Computers

Charles Babbage invented the first mechanical computer. Perhaps, this is why he got the title of "Father of Computers". Modern computers are based on his analytical machine.

Among the early inventors of computers, Lady Ada Lovelace was perhaps one of the most remarkable one. She is often credited to be the first ever programmer. Additionally, she was the first one to use an algorithm.

Similarly, the border color of elements is also modifiable.

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="border: 1px solid Violet;">First heading</h1>
    <h1 style="border: 1px solid Gray;">Second Heading</h1>
    <h1 style="border: 1px solid DodgerBlue;">Third Heading</h1>
</body>
</html>
```

This code shows the following result in a browser window where first h1 element has violet border, second h1 has gray border, and third h1 has Dodger blue border.

First heading

Second Heading

Third Heading

4.7.2 Color Values

As mentioned above, colors can also be defined using the HEX, HSL, HSLA, RGBA, and RGB values.

To use the RGB values, three separates values are required for red, green, and blue colors. These values can store a digit in the range of 0–255, where each digit represents the intensity of the color. For instance, a RGB value of (0, 255, 0) refers to the use of green color because its intensity is set to the max value for green color while red and blue are specified as having 0 values.

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="background-color: rgb(200, 100, 100);">rgb(200, 100, 100)</h1>
    <h1 style="background-color: rgb(10, 50, 90);">rgb(10, 50, 90)</h1>
    <h1 style="background-color: rgb(80, 180, 15);">rgb(80, 180, 15)</h1>
    <h1 style="background-color: rgb(240, 240, 240);">rgb(240, 240, 240)</h1>
    <h1 style="background-color: rgb(102, 202, 2);">rgb(102, 202, 2)</h1>
    <h1 style="background-color: rgb(1, 2, 3);">rgb(1, 2, 3)</h1>
    <p>RGB value example.</p>
</body>
</html>
```

This code shows the following result in a browser window where first h1 element's background is in maroon, second h1 element's background is in navy blue, third h1 element's background is in dark green, fourth h1 element's background is in light grey, fifth h1 element's background is in green, and sixth h1 element's background is in black.



Likewise, you can also use the hexadecimal format for specifying colors. The format is #rrggb which stores hexadecimal values starting from 00 to ff. For instance, #00ff00 refers to green because green's intensity is set to its maximum value in the hexadecimal notation.

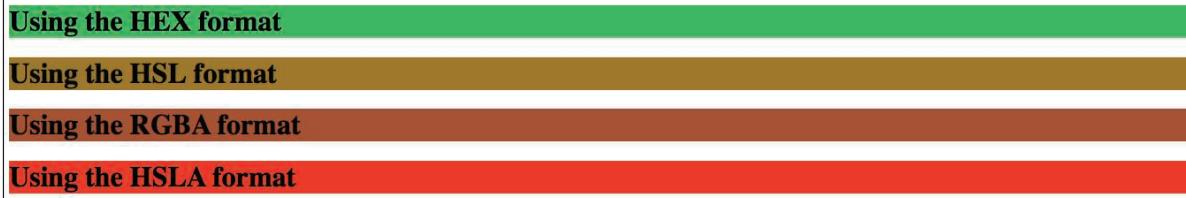
Similarly, HSL can also be used for defining colors. HSL comprises hue, saturation, and lightness. Hue is represented by a degree which falls into the range of 0 to 360 on a color wheel. Saturation and lightness are both represented by percentage values where the former applies a shade of gray while the latter applies darkness and whiteness on the colors.

We can also use the RGBA format for specifying colors. RGBA is same as RGB, except for the addition of A which refers to the alpha channel, which represents the opacity of the color.

Lastly, we have the HSLA color values which is similar to the HSL color values except for the addition of A which here also refers to the alpha channel; it represents opacity of the color. Let us see the use of all color values in the following example,

```
<!DOCTYPE html>
<html>
<body>
    <h1 style="background-color: #2bc262;">Using the HEX format</h1>
    <h1 style="background-color: hsl(40, 50%, 40%);">Using the HSL format</h1>
    <h1 style="background-color: rgba(155, 88, 61, 50);">Using the RGBA format</h1>
    <h1 style="background-color: hsla(7, 66%, 55%, 20);">Using the HSLA format</h1>
</body>
</html>
```

This code shows the following result in a browser window where first h1 element's background is in green, second h1 element's background is in light brown, third h1 element's background is in dark brown, and fourth h1 element's background is shown in red.



Are there any advantages of using color values over color names?

4.8 | Backgrounds in CSS

CSS offers several properties to edit the background of HTML elements. To add color to the background of an HTML element, you can use the background-color property as follows:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      background-color: orange;
    }
  </style>
</head>
<body>
  <h1>Background Color Property</h1>
  <p>Orange background</p>
</body>
</html>
```

This code shows the following result in a browser window where entire page's background is shown in orange color.



Similarly, you can also add an image in the background of your element. The following code shows the repetition of image below which completes the pattern for full page.

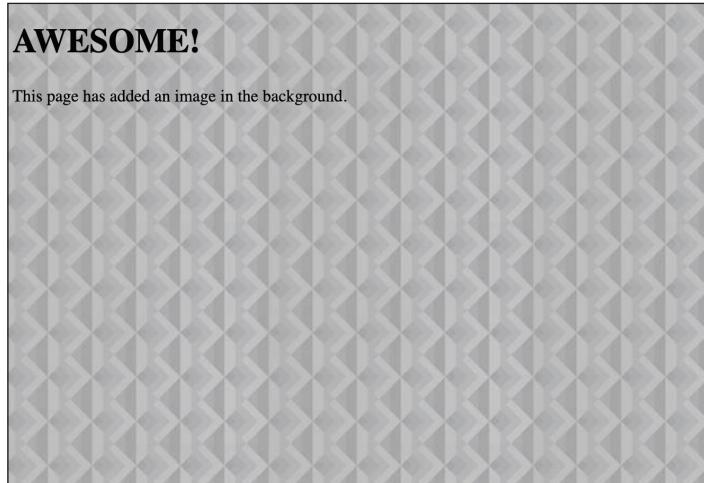


```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      background-image: url("https://www.everythingtech.co/wp-content/uploads/2019/11/verticalpattern250img.jpg");
    }
  </style>
</head>
<body>
  <h1>AWESOME!</h1>
  <p>This page has added an image in the background.</p>
</body>
</html>
```

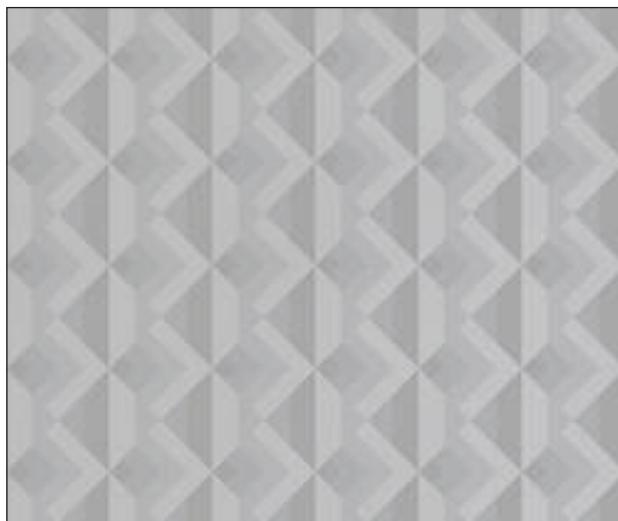
This code shows the following result in a browser window.

AWSOME!

This page has added an image in the background.

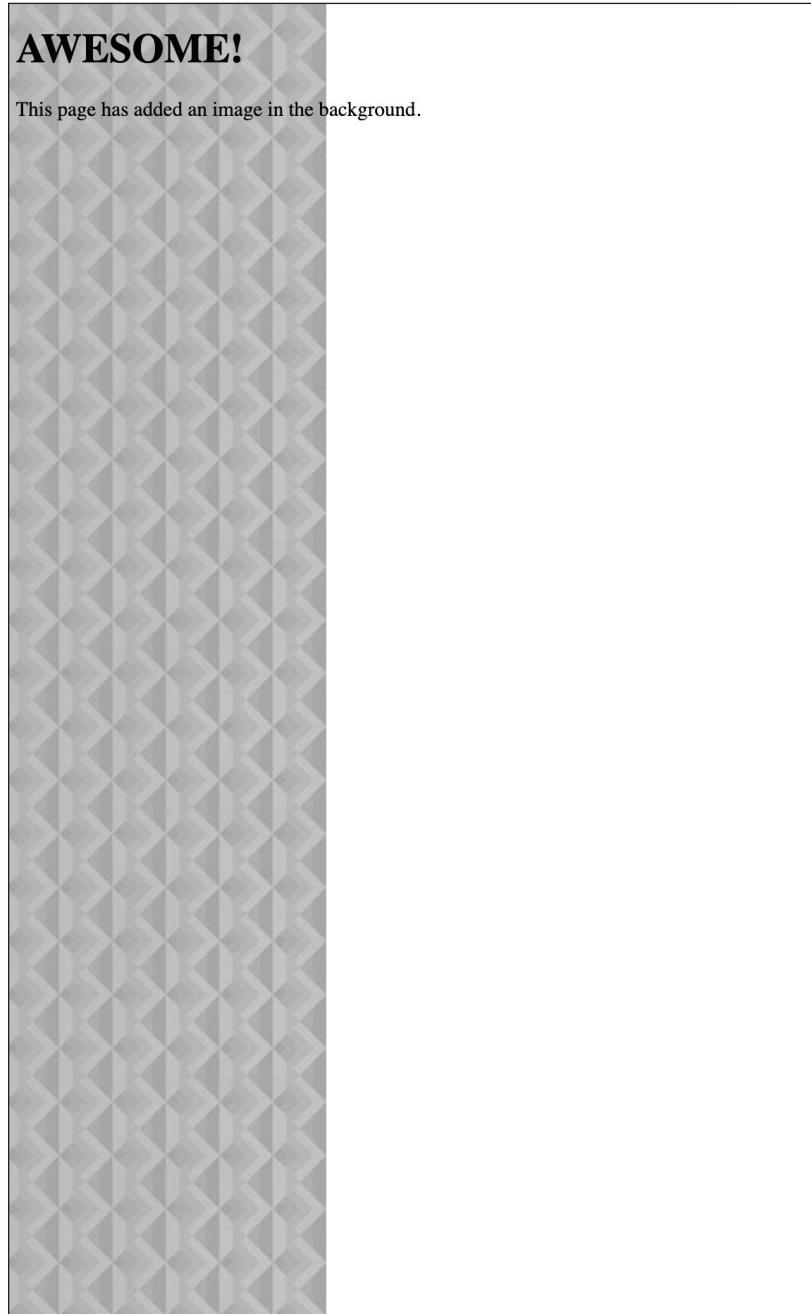


Some images look better if they are repeated either vertically or horizontally. To perform the repetition of an image vertically, use *background-repeat: repeat-y* property and for horizontal display, use *background-repeat: repeat-x* format. The following code shows the repetition of image below vertically which completes the pattern:



```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            background-image:
                url("https://www.everythingtech.co/wp-content/uploads/2019/11
/verticalpattern250img.jpg");
            background-repeat: repeat-y;
        }
    </style>
</head>
<body>
    <h1>AWSOME!</h1>
    <p>This page has added an image in the background.</p>
</body>
</html>
```

This code shows the following result in a browser window.



What if you do not want to repeat an image? Well, then you just need to add the `background-repeat` property. Sometimes, a website requires for the image to be present at all times, even while scrolling. To do this, use the `background-attachment:fixed` property. Do remember one thing: Place the image in a way that it does not affect the readability of your text.

4.9 | Setting up Height and Width of an Element

CSS can help to specify the height/width of an HTML element by using the properties "`height`" and "`width`". They can be either defined specifically in `cm`, `px`, `%` or they can be simply specified as "`auto`". The browser would automatically generate the height and width.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div {
            height: 100px;
            width: 70%;
            background-color: dodgerblue;
        }
    </style>
</head>
<body>
    <h2>Height and Width</h2>
    <p>The div element has a dodger blue color, a height of 100 px, and a width of 70%.</p>
    <div></div>
</body>
</html>
```

This code shows the following result in a browser window where the p element background is shown in Dodger blue color.

Height and Width

The div element has a dodger blue color, a height of 100 px, and a width of 70%.



When the width of the browser is lower than the element's width, then it places a horizontal scroll bar in the webpage. To address this concern, the *max-width* property is used to specify the element's maximum width.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div {
            max-width: 500px;
            height: 60px;
            background-color: dodgerblue;
        }
    </style>
</head>
<body>
    <h2>The max-width property</h2>
    <div></div>
    <p>Browser would not add any scrollbar after the resizing of the window.</p>
</body>
</html>
```

This code shows the following result in a browser window.

The max-width property

Browser would not add any scrollbar after the resizing of the window.

The browser no longer adds a scrollbar as the element repositions itself automatically.

4.10 | Box Model

A certain type of model is often discussed among web designers who work with CSS. This model, known as the *box model*, can reshape the layout and design of a webpage. It is called “box” because it is really a box which wraps around every HTML element. It comprises the following:

- 1. Margins:** A transparent property which clears the portion encircling the border.
- 2. Borders:** A non-transparent property which encircles both the content and the padding.
- 3. Padding:** A transparent property which clears the portion encircling the content.
- 4. Content:** Information for readers (i.e., text and images).

Run the below code to view a practical example of the box model.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div {
            background-color: lightgreen;
            border: 5px solid mediumseagreen;
            padding: 5px;
            margin: 5px;
        }
    </style>
</head>
<body>
    <h2>A visual demonstration of the CSS Box Model</h2>
    <div>As the name suggests, the box model is essentially a box
        which covers the content, padding, border, and margin of an element.
        In this example, we have added a 5px margin, a 5px padding, a 5px
        border, and finally a background of lightgreen for our content.</div>
</body>
</html>
```

This code shows the following result in a browser window where the div element background is in lightgreen and border is in medium sea green.

A visual demonstration of the CSS Box Model

As the name suggests, the box model is essentially a box which covers the content, padding, border, and margin of an element. In this example, we have added a 5px margin, a 5px padding, a 5px border, and finally a background of lightgreen for our content.

Understanding the box model is important for setting up the width and height of HTML elements in the web browsers. For instance, suppose we have to specify a width of 400px for an element. To specify it, we would have to stick to the following formula:

$$\text{Full Element Width} = \text{Width} + \text{Padding (left and right)} + \text{Border (left and right)} + \text{Margin (left and right)}$$

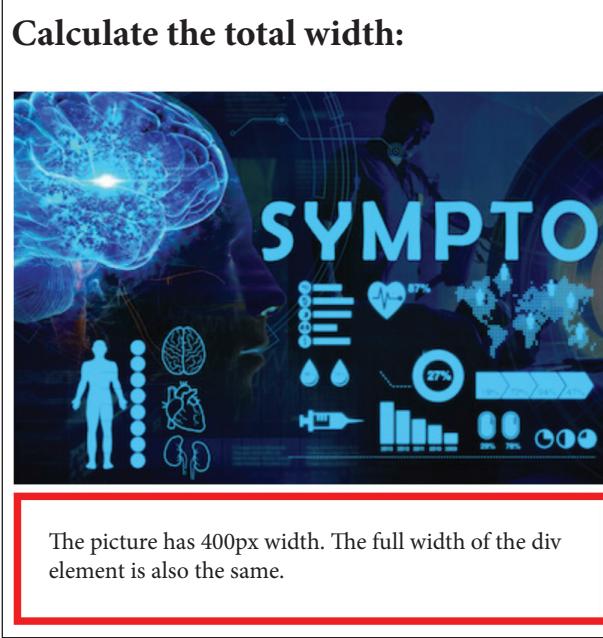
Similarly, for the complete element height, use the following formula:

Full Element Height = Height + Padding (top and bottom) + Border (top and bottom) + Margin (top and bottom)

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 350px;
    padding: 20px;
    border: 5px solid red;
    margin: 0;
}
</style>
</head>
<body>
<h2>Calculate the total width:</h2>

<div>The picture has 400px width. The full width of the div
element is also the same.</div>
</body>
</html>
```

This code shows the following result in a browser window where the div element border is in red color.



QUICK CHALLENGE

Implement multiple box models on a page and view the result in various different browsers with different resolutions.

4.11 | CSS Outline

In CSS, an outline refers to a line which is created on the sides of the element on the border's outside so the element can become more “prominent”. The outline style is defined by the *outline-style* property. It can have several values like: *hidden*, *outset*, *none*, *ridge*, *double*, *solid*, *groove*, *ridge*, *dotted*, and *dash*.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p {
            outline-color: orange;
        }

        p.a {
            outline-style: dotted;
        }

        p.b {
            outline-style: dashed;
        }

        p.c {
            outline-style: solid;
        }

        p.d {
            outline-style: double;
        }

        p.e {
            outline-style: groove;
        }

        p.f {
            outline-style: ridge;
        }

        p.g {
            outline-style: inset;
        }

        p.h {
            outline-style: outset;
        }
    </style>
</head>
<body>
    <h2>Using the style property of outline</h2>
    <p class="a">This is a dotted paragraph.</p>
    <p class="b">This is a dashed paragraph.</p>
    <p class="c">This is a solid paragraph.</p>
    <p class="d">This is a double paragraph.</p>
    <p class="e">This is a groove paragraph.</p>
    <p class="f">This is a ridge paragraph.</p>
    <p class="g">This is an inset paragraph.</p>
    <p class="h">This is an outset paragraph.</p>
</body>
</html>
```

This code shows the following result in a browser window:

Using the style property of outline

```
This is a dotted paragraph.
This is a dashed paragraph.
This is a solid paragraph.
This is a double paragraph.
This is a groove paragraph.
This is a ridge paragraph.
This is an inset paragraph.
This is an outset paragraph.
```

To specify a color for the outline, the *outline-color* property can be used. These colors can be defined in the following formats: *name*, *Hex* and *RGB* values, and invert—applying a color inversion.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p.para1 {
            border: 5px solid yellow;
            outline-style: dotted;
            outline-color: orange;
        }

        p.para2 {
            border: 5px solid yellow;
            outline-style: dashed;
            outline-color: orange;
        }

        p.para3 {
            border: 5px solid yellow;
            outline-style: inset;
            outline-color: orange;
        }
    </style>
</head>
<body>
    <h2>The outline-color Property</h2>
    <p class="para1">A solid red outline.</p>
    <p class="para2">A double green outline.</p>
    <p class="para3">An outset yellow outline.</p>
</body>
</html>
```

This code shows the following result in a browser window where all the p elements' borders are in yellow and outlines are in orange color.

The outline-color Property

```
A solid red outline.
A double green outline.
An outset yellow outline.
```

To define the outline's width, the *outline-width* property is used. It can accept the following values: a specific input (*pt*, *px*, *cm*, *em*), thick (usually 5px), medium (usually 3px), and thin (usually 1px).

```

<!DOCTYPE html>
<html>
<head>
<style>
p.para1 {
    border: 1px solid green;
    outline-style: solid;
    outline-color: orange;
    outline-width: thin;
}

p.para2 {
    border: 1px solid green;
    outline-style: solid;
    outline-color: orange;
    outline-width: medium;
}

p.para3 {
    border: 1px solid green;
    outline-style: solid;
    outline-color: orange;
    outline-width: thick;
}

p.para4 {
    border: 1px solid green;
    outline-style: solid;
    outline-color: orange;
    outline-width: 4px;
}
</style>
</head>
<body>
<h2>The outline-width Property</h2>
<p class="para1">This is an example of a thin outline.</p>
<p class="para2">This is an example of a medium outline..</p>
<p class="para3">This is an example of a thick outline.</p>
<p class="para4">This is an example of a 6px thick outline.</p>
</body>
</html>

```

This code shows the following result in a browser window.

The outline-width Property

This is an example of a thin outline.
 This is an example of a medium outline..
 This is an example of a thick outline.
 This is an example of a 6px thick outline.

For convenience, you can also use the “*outline*” property as an effective shortcut to use these properties: *outline-width*, *outline-style*, and *outline-color*.

4.12 | Text in CSS

With CSS, you can apply a number of different properties on your website’s text. For instance, to add a color to your text, you simply use the *color* property via its name or an RGB and HEX value. Likewise, to add the alignment of text such as how it needs to appear on the page, the *text-align* property is used. As an example of using both the *color* and *text-align* properties, consider the following instance:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
    text-align: center;
    color: green;
}

h2 {
    text-align: left;
    color: green;
}

h3 {
    text-align: right;
    color: green;
}
</style>
</head>
<body>
<h1>First Heading Is On The Center</h1>
<h2>Second Heading Is On The Left</h2>
<h3>Third heading Is On The Right</h3>
</body>
</html>
```

This code shows the following result in a browser window where all elements text is shown in green color.



For addition or removal of “decoration” on the text, the *text-decoration* property is used. It is sometimes used to eliminate the default underline from the hyperlinks. To check all the possible *text-decoration* values, run the following code:

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
    text-decoration: overline;
}
h2 {
    text-decoration: line-through;
}
h3 {
    text-decoration: underline;
}
a {
    text-decoration: none;
}
</style>
</head>
<body>
<h1>This is text decoration: overline</h1>
<h2>This is text decoration: line-through</h2>
<h3>This is text decoration: underline</h3>
<p>The underline is removed from the link. <a href="https://google.com">Google</a></p>
</body>
</html>
```

This code shows the following result in a browser window.

This is text decoration: overline

This is text decoration: line-through

This is text decoration: underline

The underline is removed from the link. [Google](#)

It is suggested to avoid using “*text-decoration: underline*” because it can be confusing for the visitor who may click it thinking of it as a hyperlink.

You can also apply “transformation” on the text using the *text-transform* property. Transformation means to apply the lower-case, upper-case, or capitalized format on the text.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p.lc {
            text-transform: lowercase;
        }

        p.uc {
            text-transform: uppercase;
        }

        p.cp {
            text-transform: capitalize;
        }
    </style>
</head>
<body>
    <p class="lc">This text is in lower-case.</p>
    <p class="uc">This text is in upper-case.</p>
    <p class="cp">This text is in capitalized form.</p>
</body>
</html>
```

This code shows the following result in a browser window.

this text is in lower-case.

THIS TEXT IS IN UPPER-CASE.

This Text Is In Capitalized Form.

To apply indentation on the text, the *text-indent* property is utilized. It indents the beginning line of a text. Similarly, to add space between characters, letter-spacing is used.

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
    text-indent: 120px;
    letter-spacing: 5px;
}
</style>
</head>
<body>
<p>Many people believe that Leonardo Di Caprio deserved the Oscar  

    more for Shutter Island and The Wolf of Wall Street instead of his win  

    for the Revenant. Irrespective of this sentiment, he is indeed one of  

    the best actors of this generation. His movies with Martin Scorsese  

    are a treat to watch.</p>
</body>
</html>
```

This code shows the following result in a browser window.

Many people believe that Leonardo Di Caprio deserved the Oscar more for Shutter Island and The Wolf of Wall Street instead of his win for the Revenant. Irrespective of this sentiment, he is indeed one of the best actors of this generation. His movies with Martin Scorsese are a treat to watch.

4.13 | Fonts

A plethora of font properties exist in CSS to specify the size, boldness, and font family of text. CSS groups font families into two types: generic and font. These properties are defined by using the *font-family* property. Usually, web designers add several fonts as backups because sometimes a browser may not support a specific font. When a font family has a name, which extends to more than a single word, then it is encompassed by quotation marks.

```
<!DOCTYPE html>
<html>
<head>
<style>
p.s {
    font-family: "Times New Roman", Times, serif;
}
p.ss {
    font-family: Arial, Helvetica, sans-serif;
}
</style>
</head>
<body>
<h1>Font-family Examples</h1>
<p class="s">This text is using the Times New Roman font.</p>
<p class="ss">This text is using the Arial font.</p>
</body>
</html>
```

The above code shows the following result in a browser window.

Font-family Examples

This text is using the Times New Roman font.

This text is using the Arial font.

To define the style of the text, font-style is used. It is composed of three values: *italic*, *normal*, and *oblique*.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        p.n {
            font-style: normal;
        }

        p.i {
            font-style: italic;
        }

        p.o {
            font-style: oblique;
        }
    </style>
</head>
<body>
    <p class="n">Normal Text</p>
    <p class="i">Italic Tex</p>
    <p class="o">Oblique Text</p>
</body>
</html>
```

The above code shows the following result in a browser window.

Normal Text

Italic Tex

Oblique Text

4.14 | Links in CSS

So far, we have used links in many examples. However, they are bland and unappealing. Let us make them stylish and more “functional”. All the CSS properties like *background*, *color*, *font-family* can be applied to the links. Additionally, you can add four states in links namely: *a:link*, *a:visited*, *a:hover*, and *a:active*. Let us see these states in detail:

a:link: This state is for a normal unvisited link

a:visited: This state is for a link which is visited by a user

a:hover: This state is for a link on which user has moved over mouse

a:active: This state is when user clicks on the link

Check the following example for their effect on user interaction. In this demonstration, the link initially has a yellow color. Then when it is clicked and visited, its color changes to blue. When the mouse is hovered around it, it changes into black and whenever it is clicked, the color changes to red.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        a:link {
            color: yellow;
        }

        a:visited {
            color: blue;
        }

        a:hover {
            color: black;
        }

        a:active {
            color: red;
        }
    </style>
</head>
<body>
    <p>
        <b><a href="https://google.com" target="_blank">Styling Links</a></b>
    </p>
    <p>
        <b>Reminder:</b> a:hover cannot come before a:link and a:visited in CSS for complete proper working.
    </p>
    <p>
        <b>Reminder:</b> a:active must follow a:hover property in CSS for complete working.
    </p>
</body>
</html>
```

The above code shows the following result in a browser window.

[Styling Links](#)

Reminder: a:hover cannot come before a:link and a:visited in CSS for complete proper working.

Reminder: a:active must follow a:hover property in CSS for complete working.

[Styling Links](#)

Reminder: a:hover cannot come before a:link and a:visited in CSS for complete proper working.

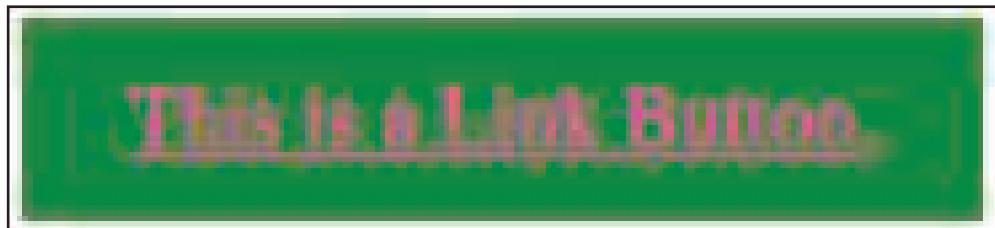
Reminder: a:active must follow a:hover property in CSS for complete working.

By using all the previous properties of CSS, let's create a link button.

```
<!DOCTYPE html>
<html>
<head>
<style>
a:link, a:visited {
    background-color: dodgerblue;
    color: hotpink;
    padding: 10px 20px;
    text-align: left;
    display: inline-block;
}

a:hover, a:active {
    background-color: GREEN;
}
</style>
</head>
<body>
<a href="google.com" target="_blank">This is a Link Button.</a>
</body>
</html>
```

This code shows the following result in a browser window where a:hover and a:active states' background color is in green and a:link and a:visited states' background color is in Dodger blue.



What is the advantage of using a:visited?

4.15 | Lists in CSS

CSS allows web designers to specify different list item markers for both unordered and ordered lists. Additionally, it facilitates developers to use an image for the marker and incorporate a background color for the list.

By using the *list-style-type* property, the marker of list items can be modified. For instance, consider the following code:

```
<!DOCTYPE html>
<html>
<head>
    <style>
        ul.b1 {
            list-style-type: circle;
        }

        ul.b2 {
            list-style-type: square;
        }

        ol.b3 {
            list-style-type: upper-roman;
        }

        ol.b4 {
            list-style-type: lower-alpha;
        }
    </style>
</head>
<body>
    <p>Using unordered lists with CSS.</p>
    <ul class="b1">
        <li>Burgers</li>
        <li>Tacos</li>
        <li>Pizza Cola</li>
    </ul>
    <ul class="b2">
        <li>Hot Dogs</li>
        <li>Sandwiches</li>
        <li>Chinese</li>
    </ul>
    <p>Using ordered lists with CSS.</p>
    <ol class="b3">
        <li>Burgers</li>
        <li>Tacos</li>
        <li>Pizza Cola</li>
    </ol>
    <ol class="b4">
        <li>Hot Dogs</li>
        <li>Sandwiches</li>
        <li>Chinese</li>
    </ol>
</body>
</html>
```

This code shows the following result in a browser window.

Using unordered lists with CSS.

- Burgers
- Tacos
- Pizza Cola

- Hot Dogs
- Sandwiches
- Chinese

Using ordered lists with CSS.

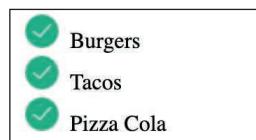
- I. Burgers
- II. Tacos
- III. Pizza Cola

- a. Hot Dogs
- b. Sandwiches
- c. Chinese

For adding an image for the list marker, use the list-style-image property as follows:

```
<!DOCTYPE html>
<html>
<head>
    <style>
        ul {
            list-style-image:
                url('https://www.everythingtech.co/wp-content/uploads/2019/11/tick24.png');
        }
    </style>
</head>
<body>
    <ul>
        <li>Burgers</li>
        <li>Tacos</li>
        <li>Pizza Cola</li>
    </ul>
</body>
</html>
```

This code shows the following result in a browser window.



4.16 | Tables in CSS

CSS can help to define different segments of HTML tables. For instance, you can use the *border* property to specify a green border for your table.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        table, th, td {
            border: 5px solid green;
        }
    </style>
</head>
<body>
    <h2>Add a border to a table:</h2>
    <table>
        <tr>
            <th>Employee Name</th>
            <th>Salary</th>
        </tr>
        <tr>
            <td>Jack</td>
            <td>60,000</td>
        </tr>
        <tr>
            <td>Bartowski</td>
            <td>70,000</td>
        </tr>
    </table>
</body>
</html>
```

This code shows the following result in a browser window where table and cell borders are in green color.

Add a border to a table:	
Employee Name	Salary
Jack	60,000
Bartowski	70,000

To collapse borders of a table and merge them in a single border, the *border-collapse* property is used.

```
<!DOCTYPE html>
<html>
<head>
<style>
table {
    border-collapse: collapse;
}

table, td, th {
    border: 5px solid green;
}
</style>
</head>
<body>
<h2>Let the borders collapse:</h2>
<table>
<tr>
    <th>Employee Name</th>
    <th>Salary</th>
</tr>
<tr>
    <td>Jack</td>
    <td>60,000</td>
</tr>
<tr>
    <td>Bartowski</td>
    <td>70,000</td>
</tr>
</table>
</body>
</html>
```

This code shows the following result in a browser window where the table and cell borders are in green color.

Let the borders collapse:	
Employee Name	Salary
Jack	60,000
Bartowski	70,000

To define the height and width of a table, CSS provides the height and width properties. We use these in the example below.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        table, td, th {
            border: 1px solid green;
        }

        table {
            border-collapse: collapse;
            width: 70%;
        }

        th {
            height: 70px;
        }
    </style>
</head>
<body>
    <h2>Example of height and width</h2>
    <table>
        <tr>
            <th>Employee Name</th>
            <th>Salary</th>
        </tr>
        <tr>
            <td>Jack</td>
            <td>60,000</td>
        </tr>
        <tr>
            <td>Bartowski</td>
            <td>70,000</td>
        </tr>
    </table>
</body>
</html>
```

This code shows the following result in a browser window.

Example of height and width

Employee Name	Salary
Jack	60,000
Bartowski	70,000

Likewise, you can use the text-align and vertical-align properties to add horizontal and vertical alignments, respectively.

4.17 | Responsiveness



With so many screen sizes, responsiveness is a necessary requirement. What if your screen is too small? In that case a normal table would be unable to display its contents. To address this issue, you can use the “*overflow-x:auto*” property; it adds responsiveness in the table.

```
<!DOCTYPE html>
<html>
<head>
<style>
table {
    border-collapse: collapse;
    width: 100%; }
th, td {
    text-align: left;
    padding: 8px; }
tr:nth-child(even) {
    background-color: dodgerblue; }
</style>
</head>
<body>
    <h2>Responsiveness in the Table</h2>
    <p>When the screen becomes small, this table can resize according and even add a scrollbar so users can read the complete information.</p>
    <div style="overflow-x: auto;">
        <table>
            <tr>
                <th>Student Name</th>
                <th>Roll Number</th>
                <th>Marks</th>
                <th>Marks</th>
                <th>Marks</th>
                <th>Marks</th>
                <th>Marks</th>
                <th>Marks</th>
                <th>Marks</th>
                <th>Marks</th></tr>
            <tr>
                <td>Tim</td>
                <td>01</td>
                <td>50</td>
                <td>60</td>
                <td>70</td>
                <td>80</td>
                <td>90</td>
                <td>60</td>
                <td>70</td>
                <td>80</td>
                <td>90</td>
                <td>60</td></tr>
            <tr>
                <td>Steve</td>
                <td>02</td>
                <td>55</td>
                <td>65</td>
                <td>75</td>
                <td>85</td>
                <td>95</td>
                <td>55</td>
                <td>65</td>
                <td>75</td>
                <td>85</td>
                <td>95</td></tr>
            <tr>
                <td>Cory</td>
                <td>03</td>
                <td>72</td>
                <td>82</td>
                <td>72</td>
                <td>82</td>
                <td>62</td>
                <td>52</td>
                <td>92</td>
                <td>62</td>
                <td>72</td>
                <td>92</td></tr>
        </table>
    </div>
</body></html>
```

This code shows the following result in a browser window.

Responsiveness in the Table

When the screen becomes small, this table can resize according and even add a scrollbar so users can read the complete information.

Student Name	Roll Number	Marks									
Tim	01	50	60	70	80	90	60	70	80	90	60
Steve	02	55	65	75	85	95	55	65	75	85	95
Cory	03	72	82	72	82	62	52	92	62	72	92

Notice, we used “*nth-child (even)*” property in the table. This property applies the background color only on the even elements in the table. Similarly, you can also add “*odd*” arrangement in the property.

QUICK CHALLENGE

List all different types of devices available in the market which have a browser to view HTML pages.

4.18 | Position Property in CSS

To define the positioning of an HTML element, the position property is used. There are five types of values in this property: *static*, *relative*, *fixed*, *absolute*, and *sticky*. After adding these properties, you can make use of the *left*, *right*, *top*, and *bottom* properties to place your elements on the webpage.

By default, all elements are static. Elements which are static are not disrupted by the *left*, *right*, *top*, and *bottom* properties. Let us see the following example,

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div.s {
            position: static;
            border: 3px solid blue;
        }
    </style>
</head>
<body>
    <h2>Static Example</h2>
    <div class="s">This is an example of static position.</div>
</body>
</html>
```

This code shows the following result in a browser window.

Static Example

This is an example of static position.

When position: relative is used with an element then it is placed relative to its position. However, if you use the *left*, *right*, *top*, and *bottom* properties with it, then the element would be disrupted and displaced from its position.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div.rel {
            position: relative;
            right: 10px;
            border: 3px solid red;
        }
    </style>
</head>
<body>
    <h2>Relative Example</h2>
    <div class="rel">This text is positioned as relative.</div>
</body>
</html>
```

This code shows the following result in a browser window.

Relative Example

This text is positioned as relative.

If you use the fixed position, then your element is placed relative with respect to the viewport. Therefore, if a visitor scrolls it, the placement of the element would not be changed. For positioning it, the *left*, *right*, *top*, and *bottom* properties are utilized.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div.fx {
            position: fixed;
            bottom: 0;
            right: 0;
            width: 200px;
            border: 5px solid blue;
        }
    </style>
</head>
<body>
    <h2>Fixed Example</h2>
    <div class="fx">This element has a fixed property.</div>
</body>
</html>
```

This code shows the following result in a browser window.

Fixed Example

An *absolute position* is that in which the element is placed with respect to the closest placed ancestor. In case, it does not have one, it assumes the document's body to be the one.

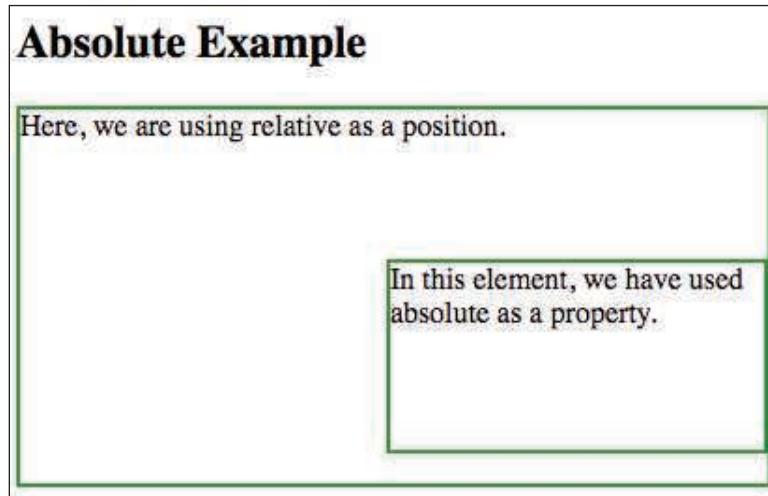
```

<!DOCTYPE html>
<html>
<head>
    <style>
        div.rel {
            position: relative;
            width: 400px;
            height: 200px;
            border: 2px solid green;
        }

        div.abs {
            position: absolute;
            top: 80px;
            right: 0;
            width: 200px;
            height: 100px;
            border: 2px solid green;
        }
    </style>
</head>
<body>
    <h2>Absolute Example</h2>
    <div class="rel">
        Here, we are using relative as a position.
        <div class="abs">In this element, we have used absolute as a property.</div>
    </div>
</body>
</html>

```

This code shows the following result in a browser window.



An HTML element whose position has been set to “sticky” is placed according to the scroll position of a user. The scroll position can either make it relative or fixed. Let us execute the following code to understand this better.

```

<!DOCTYPE html>
<html>
<head>
<style>
div.sp {
    position: sticky;
    top: 0;
    padding: 5px;
    background-color: #cae8ca;
    border: 2px solid #4CAF50;
}
</style>
</head>
<body>
<div class="sp">The Speech of Woodrow Wilson</div>
<div style="padding-bottom: 2000px">
    <p>Gentlemen of the Congress: Once more, as repeatedly before, the spokesmen of the Central Empires have indicated their desire to discuss the objects of war and the possible basis of a general peace. Parleys have been in progress at Brest-Litovsk between Russian representatives and representatives of the Central Powers to which the attention of all the belligerents has been invited for the purpose of ascertaining whether it may be possible to extend these parleys into a general conference with regard to terms of peace and settlement. The Russian representatives presented not only a perfectly definite statement of the principles upon which they would be willing to conclude peace, but also an equally definite program of the concrete application of those principles. The representatives of the Central Powers, on their part, presented an outline of settlement which, if much less definite, seemed susceptible of liberal interpretation until their specific program of practical terms was added. That program proposed no concessions at all, either to the sovereignty of Russia or to the preferences of the populations with whose fortunes it dealt, but meant, in a word, that the Central Empires were to keep every foot of territory their armed forces had occupied every province, every city, every point of vantage as a permanent addition to their territories and their power. It is a reasonable conjecture that the general principles of settlement which they at first suggested originated with the more liberal statesmen of Germany and Austria, the men who have begun to feel the force of their own peoples' thought and purpose, while the concrete terms of actual settlement came from the military leaders who have no thought but to keep what they have got. The negotiations have been broken off. The Russian representatives were sincere and in earnest. They cannot entertain such proposals of conquest and domination. The whole incident is full of significance. It is also full of perplexity. With whom are the Russian representatives dealing? For whom are the representatives of the Central Empires speaking? Are they speaking for the majorities of their respective parliaments or for the minority parties, that military and imperialistic minority which has so far dominated their whole policy and controlled the affairs of Turkey and of the Balkan States which have felt obliged to become their associates in this war?</p>
</div>
</body>
</html>

```

This code shows the following result in a browser window.

The Speech of Woodrow Wilson
<p>Gentlemen of the Congress: Once more, as repeatedly before, the spokesmen of the Central Empires have indicated their desire to discuss the objects of war and the possible basis of a general peace. Parleys have been in progress at Brest-Litovsk between Russian representatives and representatives of the Central Powers to which the attention of all the belligerents has been invited for the purpose of ascertaining whether it may be possible to extend these parleys into a general conference with regard to terms of peace and settlement. The Russian representatives presented not only a perfectly definite statement of the principles upon which they would be willing to conclude peace, but also an equally definite program of the concrete application of those principles. The representatives of the Central Powers, on their part, presented an outline of settlement which, if much less definite, seemed susceptible of liberal interpretation until their specific program of practical terms was added. That program proposed no concessions at all, either to the sovereignty of Russia or to the preferences of the populations with whose fortunes it dealt, but meant, in a word, that the Central Empires were to keep every foot of territory their armed forces had occupied-- every province, every city, every point of vantage as a permanent addition to their territories and their power. It is a reasonable conjecture that the general principles of settlement which they at first suggested originated with the more liberal statesmen of Germany and Austria, the men who have begun to feel the force of their own peoples' thought and purpose, while the concrete terms of actual settlement came from the military leaders who have no thought but to keep what they have got. The negotiations have been broken off. The Russian representatives were sincere and in earnest. They cannot entertain such proposals of conquest and domination. The whole incident is full of significance. It is also full of perplexity. With whom are the Russian representatives dealing? For whom are the representatives of the Central Empires speaking? Are they speaking for the majorities of their respective parliaments or for the minority parties, that military and imperialistic minority which has so far dominated their whole policy and controlled the affairs of Turkey and of the Balkan States which have felt obliged to become their associates in this war?</p>

4.19 | Navigation Bars

Navigation bars are one of the most important components of a website. CSS assists web designers in designing aesthetically pleasing navigation bars. A navigation bar is basically a list of links, therefore, it obviously uses HTML as a foundation. Check this basic navigation bar in HTML.

```
<!DOCTYPE html>
<html>
<body>
    <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#services">Services</a></li>
        <li><a href="#vision">Vision</a></li>
        <li><a href="#careers">Careers</a></li>
    </ul>
    <p>Since we are testing this example in an online editor, therefore we are using #. For a real website, you have to add the actual link of the website.</p>
</body>
</html>
```

This code shows the following result in a browser window.

- [Home](#)
- [Services](#)
- [Vision](#)
- [Careers](#)

Since we are testing this example in an online editor, therefore we are using #. For a real website, you have to add the actual link of the website.

But we do not use bullet points in the navigation bar. So, remove them. The margin and padding setting is done to adjust the elements to eliminate the default setting of browsers.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        ul {
            list-style-type: none;
            margin: 0;
            padding: 0;
        }
    </style>
</head>
<body>
    <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#services">Services</a></li>
        <li><a href="#vision">Vision</a></li>
        <li><a href="#careers">Careers</a></li>
    </ul>
    <p>Since, we are testing this example in an online editor, therefore we are using #. For a real website, you have to add the actual link of the website.</p>
</body>
</html>
```

This code shows the following result in a browser window.

```
Home
Services
Vision
Careers
```

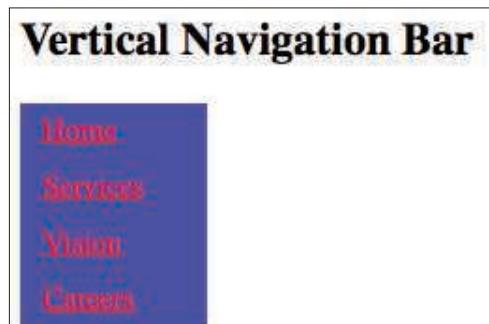
Since, we are testing this example in an online editor, therefore we are using #. For a real website, you have to add the actual link of the website.

In the above example, we generated a basic vertical navigation bar. Now, let us construct another one and add some color to its background. Additionally, we have ensured that whenever a user moves a cursor around the menu contents, the color of the bar would change according to our preference.

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
    list-style-type: none;
    padding: 0;
    width: 100px;
    background-color: blue;
}

li a {
    display: block;
    color: red;
    padding: 6px 12px;
}
/* Updating the color when the cursor hovers around it */
li a:hover {
    background-color: dodgerblue;
    color: green;
}
</style>
</head>
<body>
<h2>Vertical Navigation Bar</h2>
<ul>
    <li><a href="#home">Home</a></li>
    <li><a href="#services">Services</a></li>
    <li><a href="#vision">Vision</a></li>
    <li><a href="#careers">Careers</a></li>
</ul>
</body>
</html>
```

This code shows the following result in a browser window where ul element background is in blue.



Similarly, let us build a horizontal navigation bar. To do this, the `` elements are “floated”. This is done by using a property “`float`” so the block elements could be slided with each other. Additionally, we have added a class “`active`” in our list to show the current webpage to the user.

```

<!DOCTYPE html>
<html>
<head>
    <style>
        ul {
            list-style-type: none;
            margin: 0;
            padding: 0;
            overflow: hidden;
            background-color: brown;
        }

        li {
            float: left;
        }

        li a {
            display: block;
            color: orange;
            padding: 12px 15px;
        }

        li a:hover {
            background-color: yellow;
        }
    </style>
</head>
<body>
    <ul>
        <li><a class="active" href="#home">Home</a></li>
        <li><a href="#services">Services</a></li>
        <li><a href="#vision">Vision</a></li>
        <li><a href="#careers">Careers</a></li>
    </ul>
</body>
</html>

```

This code shows the following result in a browser window where `ul` element is shown in brown and in the second image “Careers” `li` element is hovered by a user which background is shown in yellow as set for `li a:hover` property.



Home Services Vision Careers



Home Services Vision Careers



List all the uses of navigation bar styling.

4.20 | Dropdown

When a user points at an element with a cursor, a dropdown box appears in that area of the webpage. To do this, observe the following example,

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .dropdown {
            position: relative;
            display: inline-block;
        }
        .dropdown-content {
            display: none;
            position: absolute;
            background-color: green;
            min-width: 100px;
            box-shadow: 0px 6px 12px 0px rgba(250, 250, 250, 0.2);
            padding: 12px 16px;
        }
        .dropdown:hover .dropdown-content {
            display: block;
        }
    </style>
</head>
<body>
    <h2>Basic Dropdown</h2>
    <p>Get your cursor around the following content.</p>
    <div class="dropdown">
        <span>If cursor moves here, it creates a dropdown that says:</span>
        <div class="dropdown-content">
            <p>Hi User</p>
        </div>
    </div>
</body>
</html>
```

This code shows the following result in a browser window before the cursor moves over it.

Basic Dropdown

Get your cursor around the following content.

If cursor moves here, it creates a dropdown that says:

Upon moving the cursor over to that line, you will see the following output.

Basic Dropdown

Get your cursor around the following content.

If cursor moves here, it creates a dropdown that says:

Hi User

Here we have used a “*dropdown*” class. In order to ensure that the content of the dropdown is mentioned right under the dropdown button, we used a relative position. The *.dropdown-content* class entails any content that is defined for the dropdown. By default, it is hidden on the webpage. To check its content, a user would have to move the cursor around it. The *box-shadow* property is used for giving a card-like look to the dropdown.

Let us see another example of dropdown menu. In this example, we are showing a list from our dropdown menu. It is identical to the previous example, except our use of links in a list to show options.

```
<!DOCTYPE html>
<html>
<head>
<style>
.dropbtn {
    background-color: white;
    color: gray;
    padding: 12px;
    font-size: 20px;
    border: none;
}

.dropdown {
    position: relative;
    display: inline-block;
}

.dropdown-content {
    display: none;
    position: absolute;
    background-color: gray;
    min-width: 100%;
    box-shadow: 0px 7px 14px 0px rgba(0, 0, 0, 0.9);
}

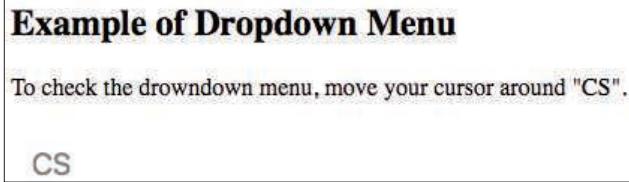
.dropdown-content a {
    color: black;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
}

.dropdown-content a:hover {
    background-color: brown
}

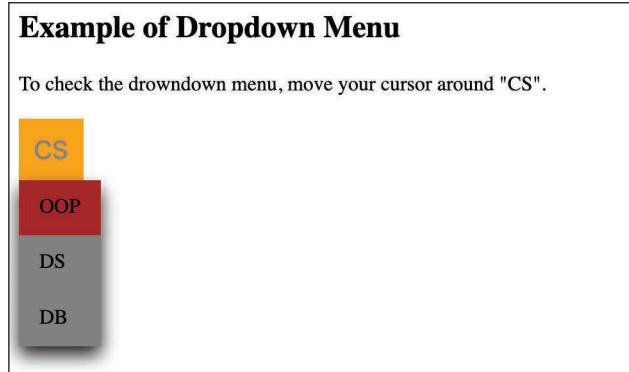
.dropdown:hover .dropdown-content {
    display: block;
}

.dropdown:hover .dropbtn {
    background-color: orange;
}
</style>
</head>
<body>
<h2>Example of Dropdown Menu</h2>
<p>To check the dropdown menu, move your cursor around "CS".</p>
<div class="dropdown">
    <button class="dropbtn">CS</button>
    <div class="dropdown-content">
        <a href="#">OOP</a><a href="#">DS</a><a href="#">DB</a>
    </div>
</div>
</body>
</html>
```

This code shows the following result in a browser window.



Upon moving over the cursor on CS, you will see the following dropdown image.



4.21 | Forms

Usually, HTML forms are styled through CSS. The input contents are specified with their types to modify their components in CSS. Consider the following example. Go through our HTML examples and you can easily see the difference in the look of the form.

```
<!DOCTYPE html>
<html>
<style>
input[type=text], select {
    width: 100%;
    padding: 10px 16px;
    margin: 6px 0;
    display: inline-block;
    border: 3px solid blue;
    border-radius: 5px;
    box-sizing: border-box;
}

input[type=submit] {
    width: 100%;
    background-color: orange;
    color: brown;
    padding: 12px 16px;
    margin: 6px 0;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

input[type=submit]:hover {
    background-color: green;
}

div {
    border-radius: 3px;
    background-color: yellow;
    padding: 15px;
}
</style>
```

```
<body>
    <h3>HTML Forms always need CSS for styling.</h3>
    <div>
        <form action="/action_page.php">
            <label for="emp">Employee Name</label><input type="text" id="emp" name="employee name" placeholder="Your name..."><label for="dept">Department</label><input type="text" id="dept" name="department" placeholder="Your department..."><label for="city">City</label><select id="city" name="city">
                <option value="Houston">Houston</option>
                <option value="Tampa">Tampa</option>
                <option value="San Francisco">San Francisco</option>
            </select><input type="submit" value="Confirm">
        </form>
    </div>
</body>
</html>
```

This code shows the following result in a browser window where the div element background is in yellow color.

HTML Forms always need CSS for styling.

Employee Name
Your name...

Department
Your department...

City
Houston

Confirm

When you hover the cursor over the “Confirm” button, you can see the background color of the button changes to green. The following image shows the button background color changed to green.

HTML Forms always need CSS for styling.

Employee Name
Your name...

Department
Your department...

City
Houston

Confirm



Can you add form validations with CSS?

Summary

CSS is a great addition to the front-end technology stack. It not only adds visually appealing design but also carries less weight in terms of page load. CSS is the default method used for web development. It is very simple to learn and use and provides a lot of styling options for all types of HTML elements.

In this chapter, we have learned the following concepts:

1. CSS and its use.
2. Styling various types of HTML elements from a label to table.

3. Id selector to style a specific element.
4. Class selection to style a group of elements.

In Chapter 5, we will learn jQuery and how to use it to add programming logic on HTML pages. jQuery is a JavaScript library, so it offers all the JavaScript features.

Multiple-Choice Questions

1. Name the text property that can be used for aligning a block elements' inline content.
 - (a) text-decoration
 - (b) text-align
 - (c) text-direction
 - (d) text-pattern
2. Which of the following properties explains whether an element is an accelerator indicator or not?
 - (a) Accelerator
 - (b) Jump-start
 - (c) Animation
 - (d) Push
3. Which one of the following CSS properties is used for padding an element?
 - (a) Padding-top
4. Which one of the following is used to describe elements in CSS?
 - (b) Padding-bottom
 - (c) Padding-left
 - (d) All of the above
5. Which one of the following CSS3 color features can be utilized as a macro for any current color?
 - (a) HSL color
 - (b) RGB color
 - (c) HSLA color
 - (d) CurrentColor Keyword

Review Questions

1. What is CSS and how it is useful?
2. What is the use of id selector?
3. What is the use of class selector?
4. How to add external style sheet?
5. Can you use HTML elements as selector to add CSS style?
6. Can you use CSS to make a page look good on all the devices?
7. What happens if multiple style sheets use the same selector?
8. Can you add animation with CSS?

Exercises

1. Create a page to show a HTML table and add CSS to make it look nicer.
2. Create a page to <div> elements. You may add nested <div> elements and add CSS to make it look like a table. See the following example and extend it further.

```
<div>
  <div>Name:</div>
  <div>MayurRamgir</div>
  <div>Address:</div>
  <div>Boston, MA, USA</div>
  <div>Course:</div>
  <div>MS Computational Science & Engineering</div>
</div>
```

3. Create a page with form elements and add CSS to make it look better.

4. Create a vertical menu bar which will stick either on the left or right side of the page.
5. Use the above created menu bar and make it movable so it will stay on the same position even after user scrolls down the page.
6. Add an image as a background to a div element and change the image source on mouse hover. In other words, when user moves a mouse pointer on that image, the image should change.
7. Add a div element and an image element. Write "My Special Div" as inline text in div like <div>My Special Div</div>. Change the text color of div element when user moves mouse over the image and change it back when user moves out of the image.
8. Add a drop down element with multiple values. Style each drop down entry differently like different font, color, size, etc.

Project Idea

Take project idea from the previous chapter where we are creating a dictionary application. Add CSS to make this application look visually appealing.

Recommended Readings

1. Eric Meyer and Estelle Weyl. 2017. *CSS: The Definitive Guide- Visual Presentation for the Web*. Shroff/O'Reilly: New Delhi
2. Thomas Powell. 2017. *HTML & CSS: The Complete Reference*. McGraw Hill Education: New York
3. Laura Lemay, Rafe Colburn, Jennifer Kyrnin. 2016. *Mastering HTML, CSS & Javascript Web Publishing*. BPB Publications: New Delhi
4. The World Wide Web Consortium (W3C) – <https://www.w3.org/Style/CSS/Overview.en.html>
5. W3School <https://www.w3schools.com/css/>
6. Mozilla Developer Network (MDN) – <https://developer.mozilla.org/en-US/docs/Web/CSS>

Introduction to jQuery

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- jQuery configuration in your application.
- Basic syntax of jQuery.
- Various events and effects.
- Animated effects for HTML elements.
- How to manipulate DOM manipulation.

5.1 | Overview of jQuery

We had given a glimpse into jQuery already in Chapter 1. By now, you must be aware of JavaScript which is the most popular and the only scripting language for the front-end development. It is event-driven interpreted language used for manipulating the HTML DOM elements. jQuery is a JavaScript library designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax. It was designed to ease JavaScript programming in the front-end. It is classified as JavaScript's light-weight library and used in more than 50% of the top 10,000 websites in the world. jQuery's vision is to enable web designers to write fewer lines of code for more functionality. Before jQuery's emergence, many routine tasks required several lines of code in JavaScript. To address this, jQuery was introduced. With jQuery, those huge lines of code were encompassed into callable methods. This means that the code of jQuery is simply JavaScript – fortunately for us, someone simplified the most common JavaScript methods through jQuery.

The library can be used for HTML, CSS, DOM manipulation, to add effects and animations, to configure event methods in HTML, and to work with AJAX. Additionally, jQuery has a variety of plugins. However, before learning jQuery, you must have a beginner level knowledge of HTML, CSS, and JavaScript.



What is the difference between JavaScript and jQuery?



5.2 | Configuration of jQuery

There are two basic approaches for incorporating the jQuery library on your website.

Approach 1: You can download it from jQuery.com. The website offers two versions:

1. **Development Version:** This is used for development and testing purposes.
2. **Production Version:** This is used for live websites. It is different from the development version due to its state of being compressed and minified, that is, its code is optimized and minimized greatly for better performance.

The library is entailed in a single JavaScript file and it is specified in the `<script>` tag of the HTML. Since it is metadata for the HTML, hence it must be placed in the `<head>` tag.

```
<head>
<script src="jQuery-3.3.1.min.js"></script>
</head>
```

Note that the downloaded file has to be put in the directory which houses the other webpages. In case, you wish to avoid downloading jQuery, then you have a better option.



What is the difference between library and framework?

Approach 2: Make use of a content delivery network (CDN) like Microsoft or Google. Most of the visitors have jQuery from Microsoft or Google when they go around websites. Therefore, it is called from the cache memory when visitors will interact with your websites. As a result, they get a quicker website loading time.

For Microsoft's jQuery integration, write the following code:

```
<head>
<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"></script>
</head>
```

For Google's jQuery integration, write the following code:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
```

When a website has several webpages, then web designers require some organization, especially if there are many jQuery functions. To do this, you can always store your functions in a different file – the file must have a *.js* extension. In this chapter, for simplicity, we are going to add them in the *<head>* tag.



Is jQuery a library or a framework?

5.3 | Syntax

In jQuery, HTML elements are “selected” or queried so that an action can be performed using them, but wait! Have not we practiced a similar practice with CSS? So what is the point of jQuery? Well, first of all, since jQuery is all JavaScript, hence you have your grip on a real programming language to modify your front-end. For example, you can use for loop in the front-end with jQuery, but the same cannot be done with CSS. Likewise, jQuery assists to add interactivity which is not possible with CSS.

The standard format for a jQuery syntax is

```
$(selector).action()
```

By breaking down this format, we have three elements. First, we have the *\$* sign that accesses jQuery. Second, we have a *selector* that searches for the specified HTML elements. Last, we have an *action* which executes a task on the selected elements. This syntax is not too dissimilar from CSS.

In our examples, we will only use methods which are defined by a document-ready event. The purpose of this approach is to block the execution of the jQuery code until the document is loaded or “ready”. This wait is recommended because it allows the complete document to be shown before jQuery performs any task. For example, jQuery can attempt to hide an HTML element which is not loaded yet on the webpage. Likewise, it can alter an image’s size before it is loaded. Additionally, this allows for JS code to be defined in the head tag, before the HTML body. Generally, this event looks like the following:

```

$(document).ready(function() {
// Any jQuery method will be defined here
});
Alternatively, there is a comparatively easier and more concise method:
$(function(){
// Any jQuery method will be defined here
});

```



Will the page throw an error if we do not use `$(document).ready` function?



5.4 | Selectors

HTML elements are manipulated through selectors in jQuery. They “select” or search for elements via their *id*, *classes*, *types*, *attributes*, or simply by their name. Selectors begin with “\$” sign and a set of parentheses “()”.



Can you have more than one “document.ready” functions?

5.4.1 Element

To find an HTML element, the element selector is used. For instance, to select a paragraph tag `<p>`, you can write: `$(“p”)`. To understand this better, observe the following:

```

<!DOCTYPE html>
<html>
    <head>
        <script
            src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
        </script>
        <script>
            $(document).ready(function() {
                $("button").click(function() {
                    $("p1").hide();
                });
            });
        </script>
    </head>
    <body>
        <h2>This Example Demonstrates Element Selector</h2>
        <p1>The first paragraph is about to disappear by using jQuery. <br></p1>
        <p2>The second paragraph is not going to disappear.</p2>
        <button id="btn">Click Here to Disappear the First Paragraph</button>
    </body>
</html>

```

Let us go through this code to understand what happens at each stage. When we open this file in a browser, the browser renders the html code and creates a Document Object Model (DOM) tree. Once the DOM tree is created, browser then creates a Cascading Style Sheet Object Model (CSSOM) tree to map CSS with HTML elements. Since browser renders HTML code, as soon as it finds `<script>` tag, it starts executing it line by line. Some browsers work differently; for example, Mozilla Firefox downloads the scripts in the background so it continues with executing HTML code.

Let us dissect our above example and see how the mentioned jQuery code gets executed.

- Step 1:** First, we have added a `<script>` tag to specify the jQuery source file. The browser then downloads this file and executes it. This gives the ability to render jQuery code. The browser can now know the jQuery tags it is going to encounter.
- Step 2:** Then it reaches to `$(document).ready(function() {` line. This line tells the browser not to execute the code from this block until the document is fully loaded. In other words, page DOM is ready for jQuery to execute.
- Step 3:** Once the execution pointer is inside the ready function, it comes to the next line which is `$("#button").click(function() {`. This code gets executed when click event occurs on the button element.
- Step 4:** When it detects the "click" event, it enters into the `$("#button").click(function() {` and finds `$("#p1").hide();`. In this, we have used a selector on "p1" tag. So, jQuery uses this to find the p1 element and hide it.

Let us see the following output of the above code. This clearly shows how easy it is to select an element and perform an action on it. The following image is taken before pressing the button.

This Example Demonstrates Element Selector

The first paragraph which is about to disappear by using jQuery.

The second paragraph which is not going to disappear. [Click Here to Disappear the First Paragraph](#)

Once the button is pressed, the first paragraph is gone (see the following image).

This Example Demonstrates Element Selector

The second paragraph which is not going to disappear. [Click Here to Disappear the First Paragraph](#)



Can you use your choice of character other than \$ on jQuery elements?

5.4.2 #id Selector

Similarly, the *id* attribute of an HTML element can also be used by jQuery to configure functionality. Note that an *id* is always unique (in a single webpage). Hence, the id selector is needed to search for a unique element. To use this selector, you must use a hash character before the element *id*. As an example, see the following code:

```
<!DOCTYPE html>
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
        <script>
            $(document).ready(function() {
                $("#button").click(function() {
                    $("#first").hide();
                });
            });
        </script>
    </head>
    <body>
        <h2>This Example Demonstrates ID Selector</h2>
        <p id="first">The first paragraph is about to disappear by using jQuery. <br>
    </p></p>
        <p>The second paragraph which is not going to disappear.</p>
        <button>Click Here to Disappear the First Paragraph</button>
    </body>
</html>
```

This code produces the following result which shows the page state before clicking the button:

This Example Demonstrate ID Selector

The first paragraph which is about to disappear by using jQuery.

The second paragraph which is not going to disappear. [Click Here to Disappear the First Paragraph](#)

Once the button is clicked, the first paragraph disappears. See the following image for better understanding.

This Example Demonstrate ID Selector

The second paragraph which is not going to disappear. [Click Here to Disappear the First Paragraph](#)

5.4.3 Class Selector

Likewise, you can use the *class* selector for choosing any HTML element with a defined class. To use this selector, you have to add a period character “.” defining the name of the class. Do remember that the difference between an HTML *id* and *class* is that multiple elements can share the same class; hence it is used for grouping. For example,

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
  </script>
    <script>
      $(document).ready(function() {
        $("button").click(function() {
          $(".hd").hide();
        });
      });
    </script>
  </head>
  <body>
    <h2 class="hd">This Example Demonstrates Class Selector</h2>
    <p1 class="hd">The first paragraph is about to disappear by using jQuery. <br>
</br></p1>
    <p2>The second paragraph is not going to disappear.</p2>
    <button>Click Here to Disappear the First Paragraph With Heading</button>
  </body>
</html>
```

This code produces the following result. This page state is of before clicking the button.

This Example Demonstrate Class Selector

The first paragraph which is about to disappear by using jQuery.

The second paragraph is not going to disappear. [Click Here to Disappear the First Paragraph With Heading](#)

After the button is clicked, the following result will be displayed. Here you can notice that the elements with the same class as “*hd*” have disappeared.



Can you unhide an element after using *.hide* on it?

5.5 | Events

jQuery is also used for manufacturing response to an “*event*” in a webpage. Whenever a visitor performs an action on an HTML page, there are pre-defined replies to provide them with a response. This pre-defined reply is called as an event. An event can be triggered when a user clicks on the *submit* button, or checks a radio-button, or even if they simply hover a mouse around an element. The triggering state of an event is often characterized by the term “*fire*”.

5.5.1 Syntax

Usually, DOM events are aided by a jQuery method. To make sure that the *<h1>* heading is clicked to trigger an event, you can write the following:

```
$(“h1”).click();
```

Afterwards, you have to specify the functionality so that a reply could be integrated whenever the event is triggered.

5.5.2 jQuery Event Methods

As we have seen, event is like a response which is triggered by a user action. jQuery provides methods to capture these user actions and reply to them.

5.5.2.1 click()

As the name suggests, this function is executed whenever an HTML element is clicked by a user. It fixes an event handler function with an HTML element. In the following example, the click method is assigned to HTML headings. When a user clicks on them, the heading gets hidden. This happens because the click event is registered for the elements used as the selector like `$(“h1, h2, h3”).click(function() {})`. This tells jQuery to execute the code in this block when a click event occurs on h1, h2, or h3 elements.

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
  </script>
  <script>
    $(document).ready(function() {
      $("h1, h2, h3").click(function() {
        $(this).hide();
      });
    });
  </script>
</head>
<body>
  <h1></h1>
  <h1>First Heading</h1>
  <h2>Second Heading</h2>
  <h3>Third Heading</h3>
</body>
</html>
```

This code shows the following result in the browser window.

First Heading

Second Heading

Third Heading

After clicking the button for the first time, the following result will be displayed.

First Heading

Second Heading

After clicking the button the second time, the following result will be displayed.

First Heading

5.5.2.2 dblclick()

To fix an event handler function with an HTML element, the `dblclick()` function is used. Let us change the example discussed in the previous subsection. In the following example, our headings are only going to be removed when the user double-clicks them.

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("h1,h2,h3").dblclick(function() {
                $(this).hide();
            });
        });
    </script>
</head>
<body>
    <h1>First Heading</h1>
    <h2>Second Heading</h2>
    <h3>Third Heading</h3>
    <p>This time, a single click is not enough. You have to double-click a heading to hide it.</p>
</body>
</html>
```

This code shows the following result in the browser window.

First Heading

Second Heading

Third Heading

This time, a single click is not enough. You have to double-click a heading to hide it.

After the button click, the following result will be displayed.

First Heading

Second Heading

This time, a single click is not enough. You have to double-click a heading to hide it.

5.5.2.3 mouseenter()

Similarly, the `mouseenter()` method is used for integrating an event handling functionality with an HTML element. Check this example, where whenever the mouse cursor enters the area of the HTML elements, an alert notifies the user.

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $(".a").mouseenter(function() {
                alert("Your cursor touched the HTML elements.");
            });
        });
    </script>
</head>
<body>
    <h1 class="a">Example of Mouseenter</h1>
    <p class="a">Take the cursor to the HTML element.</p>
</body>
</html>
```

This code shows the following result in the browser window.

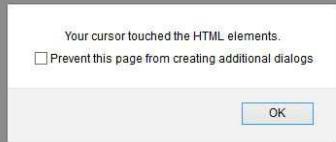
Example of Mouseenter

Take the cursor to the HTML element.

After the button is clicked, the page appears as follows.

Example of Mouseenter

Take the cursor to the HTML element.



5.5.2.4 mouseleave()

The `mouseleave()` function is triggered when the cursor of a user leaves an HTML element.

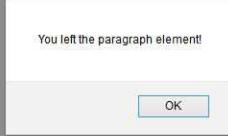
```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("#par").mouseleave(function() {
                alert("You left the paragraph element!");
            });
        });
    </script>
</head>
<body>
    <p id="par">Take the cursor to the HTML element.</p>
</body>
</html>
```

This code shows the following result in the browser window:

Take the cursor to the HTML element.

After the button click, the following result will be displayed.

Take the cursor to the HTML element.





5.6 | Effects

With jQuery, a whole variety of effects can be generated on the webpages. To enhance the user experience, we can consider adding some visual effects for actions such as button click, popup window open and close, selecting a dropdown value, show or hide various elements such as text boxes, etc.

5.6.1 `hide()` and `show()`

In jQuery, HTML elements can appear and disappear by using the `hide()` and `show()` methods. In our previous examples, we used `hide()` method to make some of the HTML elements disappear. However, we have not yet seen the `show()` method. Let us check what happens when both are used at a time:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("#a").click(function(){ //This captures the click event on element with id "a"
                $("p").hide(); //This hides all the elements p
            });
            $("#b").click(function(){//This captures the click event on element with id "b"
                $("p").show(); //This shows all the elements p
            });
        });
    </script>
</head>
<body>
    <p>If the Disappear button is clicked, then the paragraph would disappear.</p>
    <p>If the Appear button is clicked, then the paragraph would appear again.</p>
    <button id="a">Disappear</button>
    <button id="b">Appear</button>
</body>
</html>
```

This code shows the following result in the browser window.

If the Disappear button is clicked, then the paragraph would disappear.

If the Appear button is clicked, then the paragraph would appear again.

After the *Disappear* button is clicked, the following result will be displayed.

After the *Appear* button is click, the following result will be displayed.

If the Disappear button is clicked, then the paragraph would disappear.

If the Appear button is clicked, then the paragraph would appear again.

The parentheses of both `hide()` and `show()` function can take two optional arguments: *speed* and *callback*. Speed refers to the time for hiding or showing. It can be defined by “fast”, “slow”, or milliseconds (digits). On the other hand, the callback argument performs its task when the show or hide method finishes. For instance, to make sure that your HTML elements disappear after 3 seconds, you can write the following piece of code:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("button").click(function() {
                $(".a").hide(3000);
            });
        });
    </script>
</head>
<body>
    <button>Hide</button>
    <h1 class = "a">The Hidden Heading </h1>
    <p class="a">These paragraphs and the single heading will hide 3 seconds after:</p>
    <p class="a">A user clicks on the button. </p>
</body>
</html>
```

This code shows the following result in the browser window.

The Hidden Heading

These paragraphs and the single heading will hide 3 seconds after:

A user clicks on the button.

After the button click, the following result will be displayed.

**QUICK
CHALLENGE**

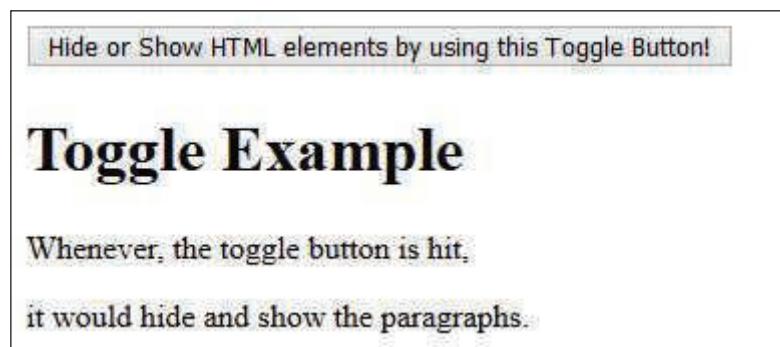
Write a code to hide and show an element in a periodic interval without any action from the user.

5.6.2 toggle()

To turn on or turn off a functionality like showing or hiding HTML elements, you can use the *toggle()* method. For example,

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("button").click(function() {
                $(".a").toggle();
            });
        });
    </script>
</head>
<body>
    <button>Hide or Show HTML elements by using this Toggle Button!</button>
    <h1 class = "a">Toggle Example </h1>
    <p class="a">Whenever, the toggle button is hit,</p>
    <p class="a">it would hide and show the paragraphs. </p>
</body>
</html>
```

This code shows the following result in the browser window.



After the button is clicked, the following result will be displayed.

Hide or Show HTML elements by using this Toggle Button!

5.6.3 Fading

For incorporating fading with an element, jQuery provides support with four methods: *fadeIn()*, *fadeOut()*, *fadeToggle()*, *fadeTo()*.

fadeIn() : This method is useful to change the opacity of an element from hidden to visible. Let us see the following example with *fadeIn()*:

```

<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#1").fadeIn(1000);
        $("#2").fadeIn(2000);
        $("#3").fadeIn(3000);
    });
});
</script>
</head>
<body>
<p>Applying fadeIn effect on each box with different timing.</p>
<button>Hit the button to get the fading effect.</button><br><br>
<div id="1" style="width:40px;height:60px;display:none;background-color:brown;">
</div><br>
<div id="2" style="width:40px;height:60px;display:none;background-color:yellow;">
</div><br>
<div id="3" style="width:40px;height:60px;display:none;background-color:orange;">
</div>
</body>
</html>

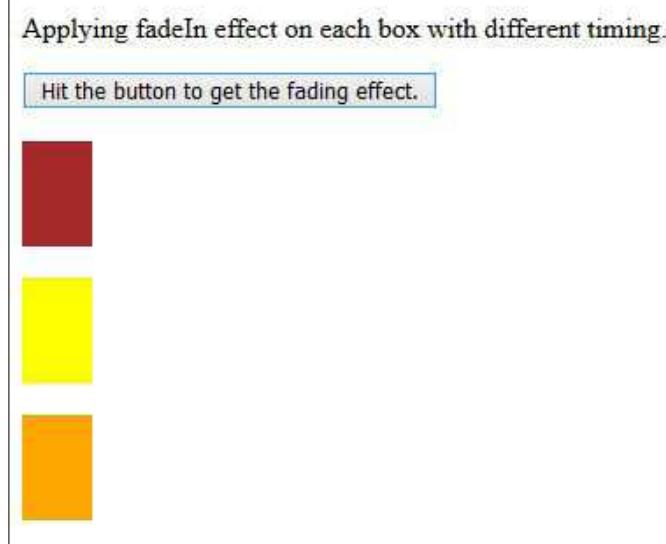
```

This code shows the following result in the browser window.

Applying fadeIn effect on each box with different timing.

Hit the button to get the fading effect.

After the button click, the following result will be displayed.



**QUICK
CHALLENGE**

Define `fadeout()`, `fadeToggle()`, and `fadeTo()` in detail. Give examples of each.

5.6.4 Sliding

jQuery is often known for its beautiful and interactive sliders. It offers three basic slide methods: `slideDown()`, `slideUp()`, and `slideToggle()`.

- 1. `slideDown()`:** This method is useful to reveal an element slowly, from top to bottom. It animates the height of the element to reveal the lower part of the element.

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("#flip").click(function(){
                $("#panel").slideDown("fast");
            });
        });
    </script>
    <style>
        #panel, #flip {
            padding: 10px;
            text-align: left;
            background-color: dodgerblue;
            border: solid 5px purple;
        }
        #panel {
            padding: 60px;
            display: none;
        }
    </style>
</head>
<body>
    <div id="flip">When this area is clicked, a panel emerges.</div>
    <div id="panel">The panel greets the user with Merry Christmas!</div>
</body>
</html>
```

This code shows the following result in the browser window.

When this area is clicked, a panel emerges.

After the button is clicked, the following result will be displayed.

When this area is clicked, a panel emerges.

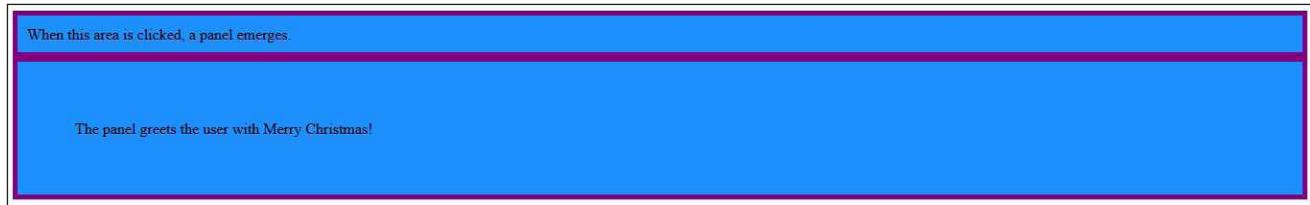
The panel greets the user with Merry Christmas!

Similarly, to slide up an HTML element, we can use the `slideUp()` method.

2. **slideUp():** This method is useful to sliding up an element from bottom to top, which gives animated effect of concealing. We show this using the example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("#flip").click(function() {
                $("#panel").slideUp("fast");
            });
        });
    </script>
    <style>
        #panel, #flip {
            padding: 10px;
            text-align: left;
            background-color: dodgerblue;
            border: solid 5px purple;
        }
        #panel {
            padding: 60px;
        }
    </style>
</head>
<body>
    <div id="flip">When this area is clicked, a panel emerges.</div>
    <div id="panel">The panel greets the user with Merry Christmas!</div>
</body>
</html>
```

This code shows the following result in the browser window.



After the button is clicked, the following result will be displayed.



5.6.4.1 slideToggle()

To enable or disable sliding, you can use the `slideToggle()` method. This method is useful to enable or disable sliding. In other words, if the element at its full height then it conceal it, and if it is concealed then it reveals the element. Let us see the example:

```

<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function() {
  $("#flip").click(function() {
    $("#panel").slideToggle("fast");
  });
});
</script>
<style>
#panel, #flip {
  padding: 10px;
  text-align: left;
  background-color: dodgerblue;
  border: solid 5px purple;
}
#panel {
  padding: 60px;
  display: none;
}
</style>
</head>
<body>
  <div id="flip">When this area is clicked, a panel emerges.</div>
  <div id="panel">The panel greets the user with Merry Christmas!</div>
</body>
</html>

```

This code shows the following result in the browser window.

When this area is clicked, a panel emerges

After the button is clicked, the following result will be displayed.

When this area is clicked, a panel emerges

The panel greets the user with Merry Christmas!

5.6.5 Animation

It is also possible to add movements via animations in jQuery. To do this, you require the `animate()` method. The basic syntax of the `animate()` method is the following:

```
$(selector).animate({params}, speed, callback);
```

Params is short for parameter and references to CSS properties that require animation. Speed refers to the time period for animation – “fast”, “slow”, and milliseconds are arguments. The callback parameter runs after the completion of the animation. We would read more on *Callback* in a later section.

Note that by default, HTML elements are static which means it is not possible to move them. To move them, you have to adjust the CSS property of that element to absolute, fixed, or relative.

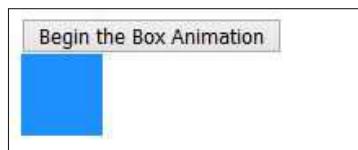
For a simple animation, check the following example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("button").click(function() {
                $("div").animate({left: '500px'});
            });
        });
    </script>
</head>
<body>
    <button>Move the box!</button>
    <p></p>
    <div style="background:dodgerblue;height:50px;width:50px;position:fixed;"></div>
</body>
</html>
```

Let us move towards more advanced animations where we add multiple properties to our box.

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("button").click(function() {
                $("div").animate({
                    left: '500px',
                    opacity: '0.2',
                    height: '100px',
                    width: '100px'
                });
            });
        });
    </script>
</head>
<body>
    <button>Begin the Box Animation</button>
    <div style="background:dodgerblue;height:50px;width:50px;position:fixed;"></div>
</body>
</html>
```

This code shows the following result in the browser window.



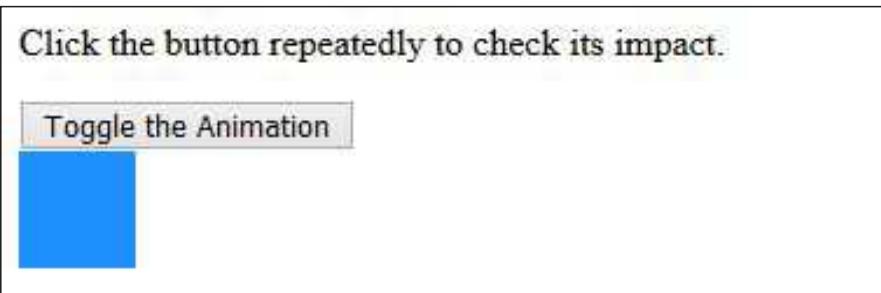
After the button is clicked, the following result will be displayed.



You can also use the *hide*, *show*, and *toggle* values in the animate method. For instance, consider the following example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function(){
            $("button").click(function(){
                $("div").animate({
                    width: 'toggle'
                });
            });
        });
    </script>
</head>
<body>
    <p>Click the button repeatedly to check its impact.</p>
    <button>Toggle the Animation</button>
    <div style="background: dodgerblue; height: 50px; width: 50px; position: absolute;">
    </div>
</body>
</html>
```

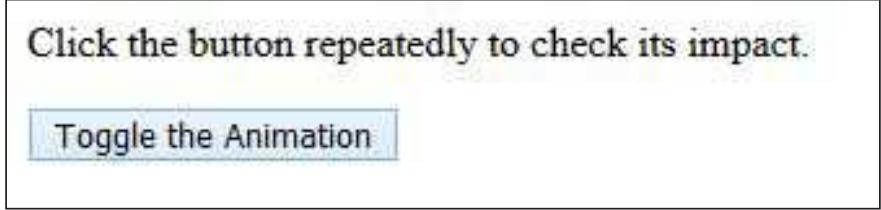
This code shows the following result in the browser window.



Click the button repeatedly to check its impact.

Toggle the Animation

After the button is clicked, the following result will be displayed.



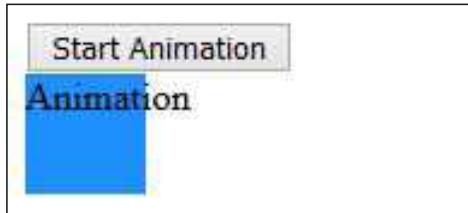
Click the button repeatedly to check its impact.

Toggle the Animation

jQuery supports queue functionality. Hence, if multiple calls are written in the `animate()` method, then jQuery builds a queue with them, that is, calls `animate` step-by-step like a real-world queue. For example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        var div = $("div");
        div.animate({height: '50px', opacity: '0.3'}, "slow");
        div.animate({width: '300px', opacity: '0.7'}, "fast");
        div.animate({height: '50px', opacity: '0.3'}, "fast");
        div.animate({width: '300px', opacity: '0.7'}, "slow");
        div.animate({fontSize: '2em'}, "slow");
    });
});
</script>
</head>
<body>
<button>Start Animation</button>
<div style="background:dodgerblue;height:50px;width:50px;position:absolute;">
Animation</div>
</body></html>
```

This code shows the following result in the browser window.



After the button is clicked, the following result will be displayed.



Note that this time we also added text in our animation and used it in the queue to animate it as well.

5.6.6 jQuery Stop Animations

When an animation or effect is still in process, then the `stop()` method can halt it before its completion. This method is applicable to *fading*, *sliding*, *custom animations*, and any effect functions. The basic format of the method is as follows:

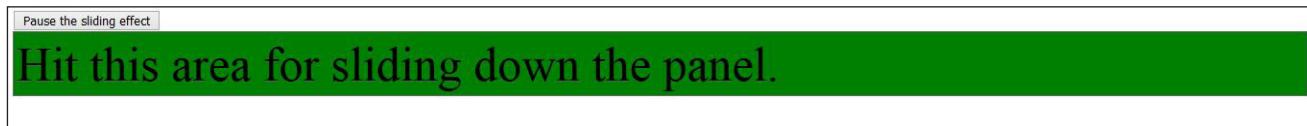
```
$(selector).stop(stopAll, goToEnd);
```

These parameters are optional. The “`stopAll`” parameter is used to specify if the queue in the animation requires to be cleared. By default, it is `false`; therefore, only the animation which is active is blocked, thereby permitting other animations in the queue to be executed. The “`goToEnd`” parameter defines if the current animation should be finished quickly. By default, it is `false`.

The `stop()` method stops any current animation, depending upon the element which is selected. For instance, consider the following example.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function() {
  $("#flip").click(function() {
    $("#panel").slideDown(2000);
  });
  $("#stop").click(function() {
    $("#panel").stop();
  });
});
</script>
<style>
#panel, #flip {
  padding: 3px;
  font-size: 50px;
  text-align: left;
  background-color: green;
  color: black;
  border: solid 1px #666;
}
#panel {
  padding: 20px;
  display: none;
}
</style>
</head>
<body>
<button id="stop">Pause the sliding effect</button>
<div id="flip">Hit this area for sliding down the panel.</div>
<div id="panel">Hello User! </div>
</body>
</html>
```

This code shows the following result in the browser window.



5.6.7 Callback Function

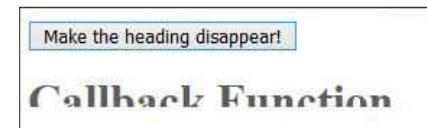
By default, JavaScript functions are executed and processed on a line-by-line basis. Sometimes, this is problematic as the next line of code is executed before the completion of an effect. As a consequence, your webpage may face errors. In order to avoid this, we use a callback function. When the current effect is completed, then the callback function is executed. For instance, consider the following example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function() {
    $("button").click(function() {
        $("h1").hide(3000, function() {
            alert("The heading has disappeared.");
        });
    });
    </script>
</head>
<body>
    <button>Make the heading disappear!</button>
    <h1>Callback Function</h1>
</body></html>
```

This code shows the following result in the browser window.



After the button click, the following result will be displayed.

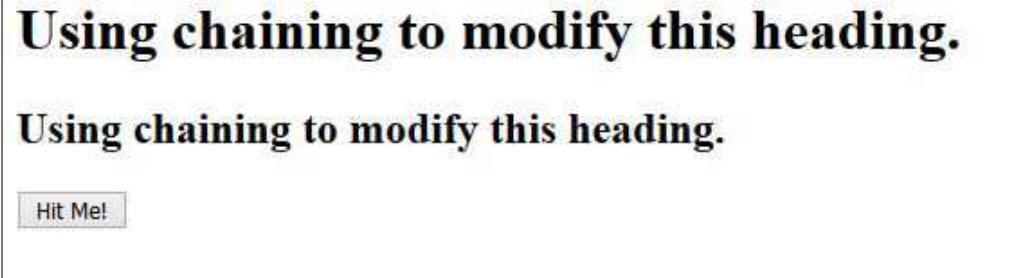


5.6.8 Chaining

As the name suggests, chaining is the “chain” of multiple jQuery methods. Chaining in jQuery refers to the execution of more than a single jQuery method with a single element in one statement. This is useful for browsers because they do not have to search for an element repeatedly. For chaining, you just have to append the prior action to the current action. For instance,

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function() {
    $("button").click(function() {
        $("h").css("color", "blue").slideUp(2000).slideDown(2000);
    });
});
</script>
</head>
<body>
    <h>Using chaining to modify this heading.</h>
    <h>Using chaining to modify this heading.</h>
    <button>Hit Me!</button>
</body>
</html>
```

This code shows the following result in the browser window.

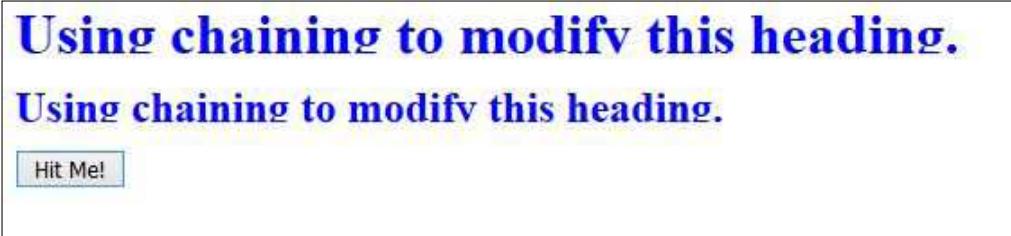


Using chaining to modify this heading.

Using chaining to modify this heading.

Hit Me!

After the button is clicked, the following result will be displayed.



Using chaining to modify this heading.

Using chaining to modify this heading.

Hit Me!

5.7 | Working with HTML

jQuery contains powerful methods for changing and manipulating HTML elements and attributes. In this section, we will cover various such methods and explain with examples the use of those.

5.7.1 DOM Manipulation

jQuery allows users to modify Document Object Model (DOM). DOM is a standard which is used to access HTML and XML documents. Properties and methods define DOM's programming interface. Property is something the element has and method is something the element can do. There are various DOM properties around but following are some of the more significant ones. Let us say "n" is a node object.

1. **n.innerHTML:** This property refers to the inner text value of an HTML element "n". However, note that it parses content as HTML so it takes longer to get value compared to nodeValue property. For example, say "n" refers to <a> element like Mayur Ramgir. In this case, n.innerHTML will give "Mayur Ramgir".
2. **n.nodeName:** This is very straightforward property as it gives the name of the node "n".
3. **n.nodeValue:** This property is similar to innerHTML in a sense as it gives the inner text of node "n". However, a major difference is it does not parse HTML and gets the value in text format. Hence, it is faster than innerHTML.
4. **n.parentNode:** This property gives access to the parent node of "n".
5. **n.childNodes:** This property gives access to the child node of "n".
6. **n.attributes:** This property gives all the attributes of node "n".

5.7.2 Get Content and Attributes

The following are three basic and easy-to-use methods in jQuery for DOM manipulation:

1. The `text()` method is used to assign or return any piece of text for the selected HTML elements.
2. The `html()` method is used to assign or return the content for the selected elements (markup included).
3. The `val()` method is used to assign or return any value from the fields in the HTML forms.

To see the `text()` and `html()` methods, consider the following example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("#b1").click(function() {
                alert("Text: " + $("#par").text());
            });
            $("#b2").click(function() {
                alert("HTML: " + $("#par").html());
            });
        });
    </script>
</head>
<body>
    <p id="par">This is line is part of a paragraph <b>and contains bold text.</b> </p>
    <button id="b1">Display Text</button>
    <button id="b2">Display HTML code</button>
</body>
</html>
```

This code shows the following result in the browser window.

This is line is part of a paragraph and contains bold text.

Display Text

Display HTML code

After the button is clicked, the following result will be displayed.

This is line is part of a paragraph and contains bold text.

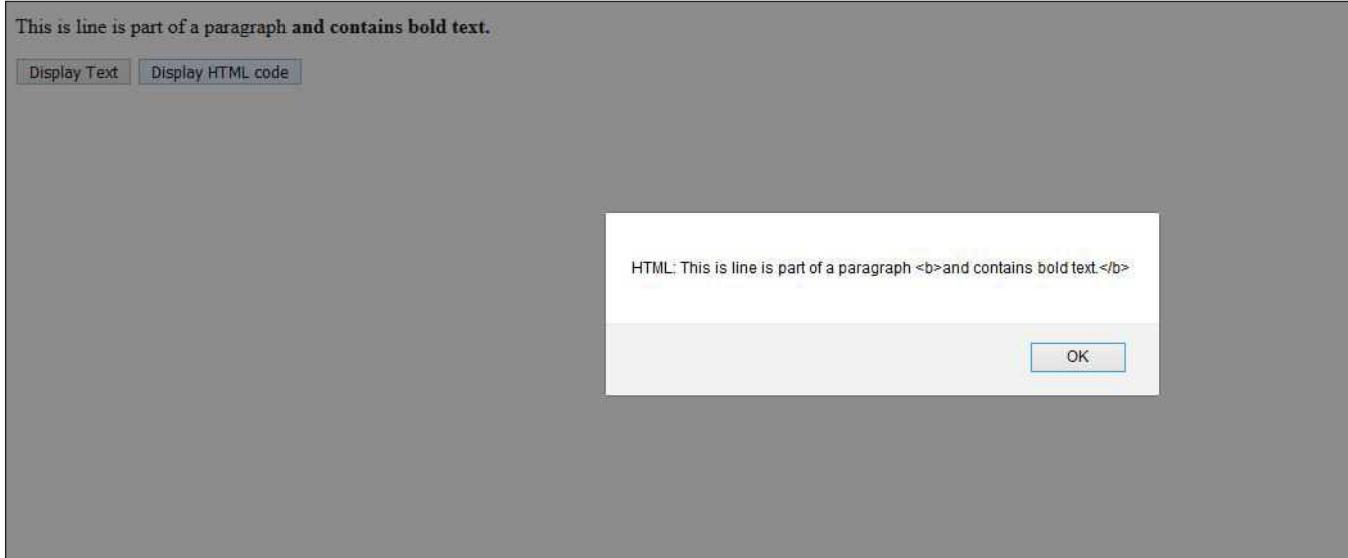
Display Text

Display HTML code

Text: This is line is part of a paragraph and contains bold text.

OK

After the button is clicked again, the following result will be displayed.



Similarly, to check the use of `val()`, consider the following:

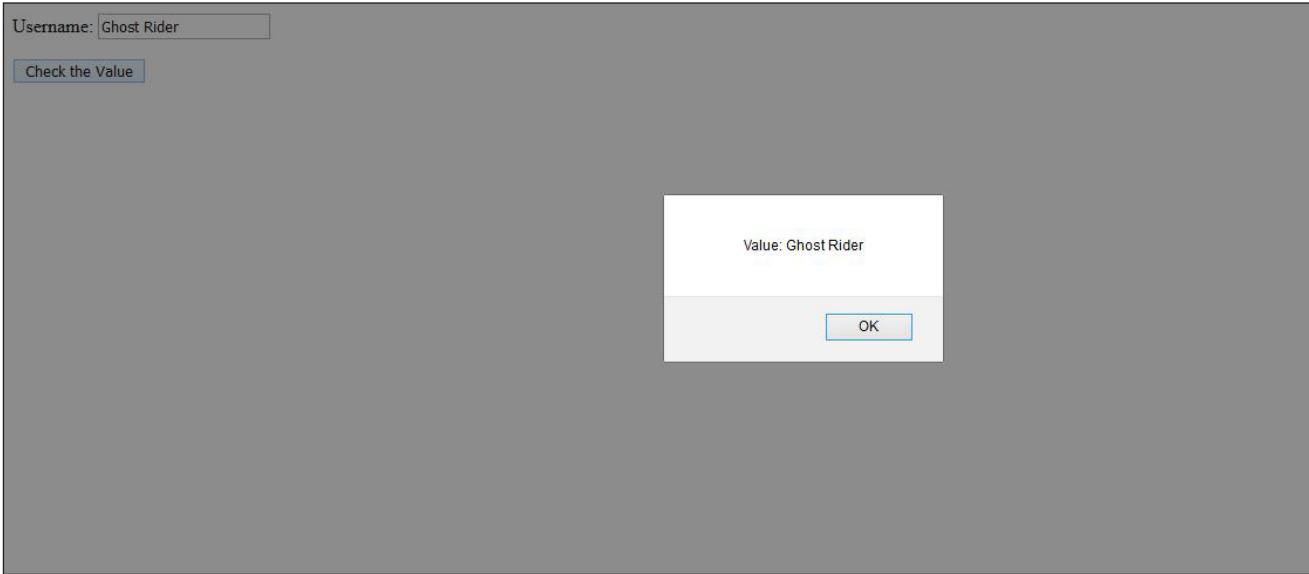
```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("#button1").click(function() {
                alert("Value: " + $("#txt").val());
            });
        });
    </script>
</head>
<body>
    <p>Username: <input type="text" id="txt" value="Ghost Rider"></p>
    <button id ="button1">Check the Value</button>
</body>
</html>
```

This code shows the following result in the browser window.

Username:

Check the Value

After the button is clicked, the following result will be displayed.



5.7.2.1 Get Attributes

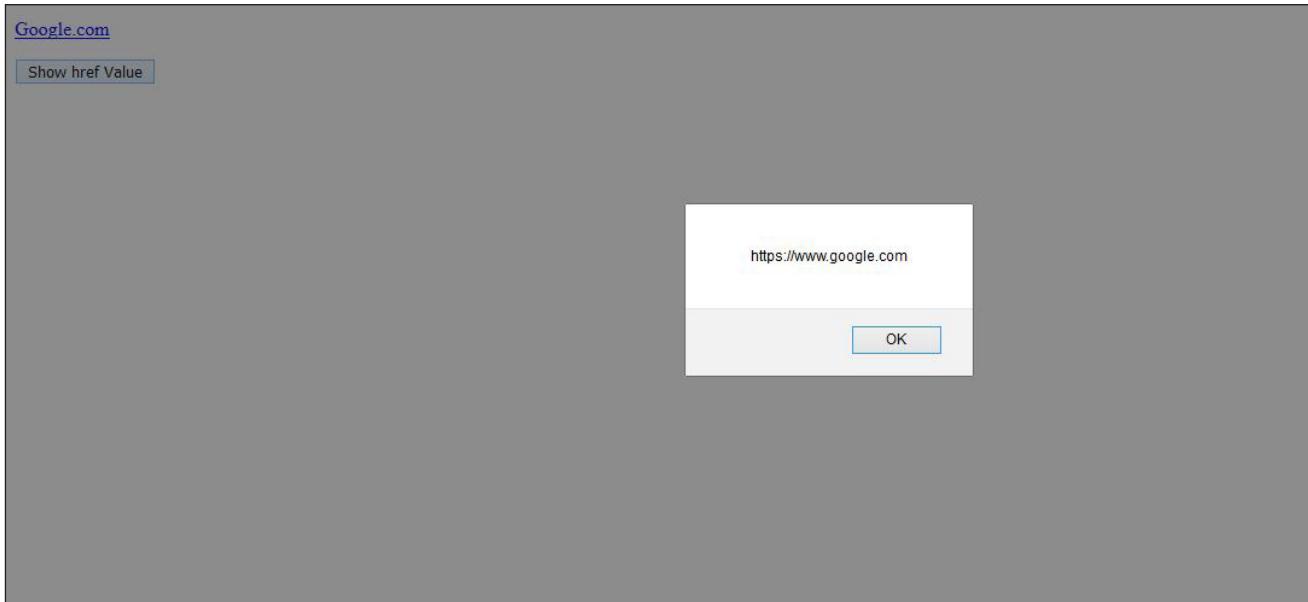
To receive the value of HTML attributes, jQuery provides a method, `attr()`. Go through the following example in which the value of the `href` is displayed by the jQuery function.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        alert($("#go").attr("href"));
    });
});
</script>
</head>
<body>
<p><a href="https://www.google.com" id="go">Google.com</a></p>
<button>Show href Value</button>
</body>
</html>
```

This code shows the following result in the browser window.



After the button is clicked, the following result will be displayed.



5.7.3 Set Content and Attributes

The above-mentioned methods can also be used for setting content. For instance, check the following example in which we have used set content with the jQuery `html()`, `val()`, and `text()` methods.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function() {
    $("#b1").click(function() {
        $("#par1").text("Hi User!");
    });
    $("#b2").click(function() {
        $("#par2").html("<b>Hi User!</b>");
    });
    $("#b3").click(function() {
        $("#par3").val("Shane");
    });
});
</script>
</head>
<body>
<p id="par1">This is our first paragraph.</p>
<p id="par2">This is our second paragraph.</p>
<p>Input field: <input type="text" id="par3" value="Rick"></p>
<button id="b1">Setting any textual information</button>
<button id="b2">Setting any HTML data</button>
<button id="b3">Setting any value</button>
</body>
</html>
```

This code shows the following result in the browser window.

This is our first paragraph.

This is our second paragraph.

Input field:

Setting any textual information Setting any HTML data Setting any value

After the button is clicked, the following result will be displayed.

Hi User!

This is our second paragraph.

Input field:

Setting any textual information Setting any HTML data Setting any value

After the button is clicked again, the following result will be displayed.

Hi User!

Hi User!

Input field:

Setting any textual information Setting any HTML data Setting any value

After the button is clicked once again, the following result will be displayed.

Hi User!

Hi User!

Input field:

Setting any textual information Setting any HTML data Setting any value

5.7.3.1 Set Attributes

To change or modify the values of an attribute, we can use the `attr()` method. Check the following example in which we have modified the value of `href` to redirect a user to another website. By now, you must have realized how jQuery can help you to gain a wider range of control and manipulation on the front-end coding of HTML.

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function(){
            $("#b1").click(function(){
                $("#w3s").attr("href", "https://www.reddit.com");
            });
        });
    </script>
</head>
<body>
    <p id="par1"><a href="https://www.google.com" id="w3s">Google.com</a></p>
    <button id="b1">Click to change the href value</button>
    <p>After clicking the link, place your cursor on the hyperlink and check the link address or click it to go to other website.</p>
</body>
</html>
```

This code shows the following result in the browser window.

The screenshot shows a browser window with the following content:

- A heading "Google.com" with a blue underline.
- A button labeled "Click to change the href value".
- A paragraph of text: "After clicking the link, place your cursor on the hyperlink and check the link address or click it to go to other website."

5.7.4 Addition of Elements

For the addition of fresh contents in HTML elements, jQuery provides four methods: `append()`, `prepend()`, `after()`, and `before()`.

5.7.4.1 append()

Whenever you want to insert any piece of content at the extreme end of your element, use the `append()` method. In the following example, we have appended text at the end of the paragraphs as well attached one more list item to the list.

```

<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("#b1").click(function(){
        $("p").append(" <b>New Content</b>.");
    });
    $("#b2").click(function(){
        $("ul").append("<li>Soup</li>");
    });
});
</script>
</head>
<body>
<p>This is the first paragraph.</p>
<p>This is the second paragraph.</p>
<ul>
<li>Coffee</li>
<li>Tea</li>
<li>Water</li>
</ul>
<button id="b1">Append content at the end of paragraph.</button>
<button id="b2">Append item at the end of the list.</button>
</body>
</html>

```

This code shows the following result in the browser window.

This is the first paragraph.
 This is the second paragraph.
 • Coffee
 • Tea
 • Water

[Append content at the end of paragraph.](#) [Append item at the end of the list.](#)

After the button is clicked, the following result will be displayed.

This is the first paragraph. **New Content.**
 This is the second paragraph. **New Content.**
 • Coffee
 • Tea
 • Water

[Append content at the end of paragraph.](#) [Append item at the end of the list.](#)

After the button is clicked again, the following result will be displayed.

```
This is the first paragraph. New Content.  
This is the second paragraph. New Content.  
• Coffee  
• Tea  
• Water  
• Soup  
  
Append content at the end of paragraph. Append item at the end of the list.
```

5.7.4.2 prepend()

Similarly, in order to add content at the starting of an element, we can use the `prepend()` method.

```
<!DOCTYPE html>  
<html>  
<head>  
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">  
</script>  
    <script>  
        $(document).ready(function() {  
            $("#b1").click(function() {  
                $("p").prepend("<b>New Content</b>. ");  
            });  
            $("#b2").click(function() {  
                $("ol").prepend("<li>Nuggets</li>");  
            });  
        } );  
    </script>  
</head>  
<body>  
    <p>This is the first paragraph.</p>  
    <p>This is the second paragraph.</p>  
    <ol>  
        <li>Pizza</li>  
        <li>Burger</li>  
        <li>Sandwich</li>  
    </ol>  
    <button id="b1">Prepend content at the end of paragraph.</button>  
    <button id="b2">Prepend item at the end of the list.</button>  
</body>  
</html>
```

This code shows the following result in the browser window.

```
This is the first paragraph.  
This is the second paragraph.  
1. Pizza  
2. Burger  
3. Sandwich  
  
Prepend content at the end of paragraph. Prepend item at the end of the list.
```

After the button is clicked, the following result will be displayed.

```
New Content. This is the first paragraph.  
New Content. This is the second paragraph.  
1. Pizza  
2. Burger  
3. Sandwich
```

Prepend content at the end of paragraph. **Prepend item at the end of the list.**

After the button is clicked again, the following result will be displayed.

```
New Content. This is the first paragraph.  
New Content. This is the second paragraph.  
1. Nuggets  
2. Pizza  
3. Burger  
4. Sandwich
```

Prepend content at the end of paragraph. **Prepend item at the end of the list.**

So far, we have written some basic examples with `append()` and `prepend()` but both of these methods are powerful enough to take elements as parameters. New HTML elements such as headings can be created from scratch by making use of jQuery. For instance, in the following example we have generated elements by using jQuery, HTML, and DOM. The `append()` method is used to add new elements.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
function appTxt() {
    var t1 = "<p>Using HTML to generate text</p>";
    var t2 = $("<p></p>").text("Using jQuery to generate text.");
    var t3 = document.createElement("p");
    t3.innerHTML = "Some Content.";
    $("body").append(t1, t2, t3);
}
</script>
</head>
<body>
<button onclick="appTxt ()">Append text</button>
</body>
</html>
```

This code shows the following result in the browser window.

Append text

After the button is clicked, the following result will be displayed.

Append text
<p>Using HTML to generate text</p> <p>Using jQuery to generate text.</p> <p>Some Content.</p>

5.7.4.3 after() and before()

The `after()` and `before()` methods are used to add information after or before the specified elements. For example, to add content before and after an image, we can write the following code:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function(){
            $("#b1").click(function(){
                $("img").before("<b>Adding text before the image.</b>");
            });
            $("#b2").click(function(){
                $("img").after("Adding text after the image.");
            });
        });
    </script>
</head>
<body>
    
    <br><br>
    <button id="b1">Adding text before the image.</button>
    <button id="b2">Adding text after the image.</button>
</body>
</html>
```

This code shows the following result in the browser window.


<input type="button" value="Adding text before the image."/> <input type="button" value="Adding text after the image."/>

After the button is clicked, the following result will be displayed.



After the button is clicked again, the following result will be displayed.



5.7.5 Deleting HTML Elements

jQuery allows us to easily delete elements from the HTML. For this, two main methods used are `remove()` and `empty()`.

5.7.5.1 `remove()`

To delete any selected single element or set of elements, jQuery provides `remove()` method. It also deletes the child elements of those elements. In the following example, we designed a `div` element and called the `remove()` method with it.

```
<!DOCTYPE html>
<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
  <script>
  $(document).ready(function() {
    $("button").click(function() {
      $("#d").remove();
    });
  });
  </script>
</head>
<body>
  <div id="d" style="height:500px;width:500px;border:3px solid green;
background-color:orange;">
    We are using div element in our example.
    <p>A button is used for deletion.</p>
    <p>The button calls out the remove() method to delete the element.</p>
  </div>
  <br>
  <button>Delete the div element from the page.</button>
</body>
</html>
```

This code shows the following result in a browser window which has orange background color and green border.

We are using div element in our example.

A button is used for deletion.

The button calls out the remove() method to delete the element.

Delete the div element from the page.

After the button is clicked, the following result will be displayed, where the entire div is deleted.

Delete the div element from the page.

The remove() method can also take one parameter for filtering of elements. For instance, to make sure that all the paragraphs with "b" class are deleted, we can write the following.

```

<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").remove(".b");
    });
});
</script>
<style>
.b {
    color: dodgerblue;
    font-size: 40px;
}
</style>
</head>
<body>
<p class ="a">This is the first paragraph.</p>
<p class="b">This is the second paragraph.</p>
<p class="b">This is the third paragraph.</p>
<button>Remove all p elements with class="test"</button>
</body>
</html>

```

This code shows the following result in a browser window where the text of p element which got class b is shown in dodgerblue.

This is the first paragraph.

This is the second paragraph.

This is the third paragraph.

Remove all p elements with class="test"

After the button is clicked, the following result will be displayed.

This is the first paragraph.

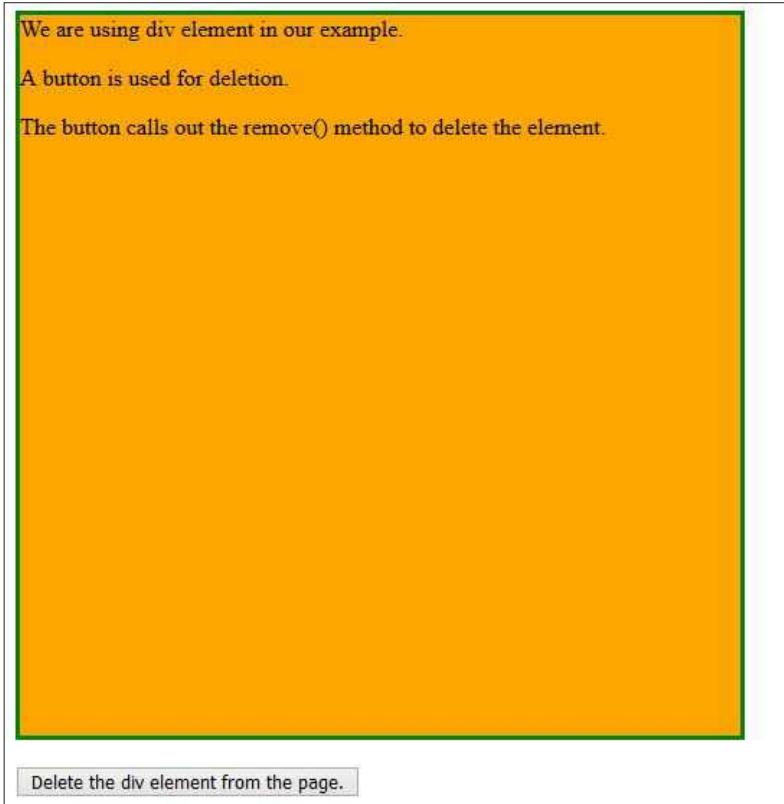
Remove all p elements with class="test"

5.7.5.2 empty()

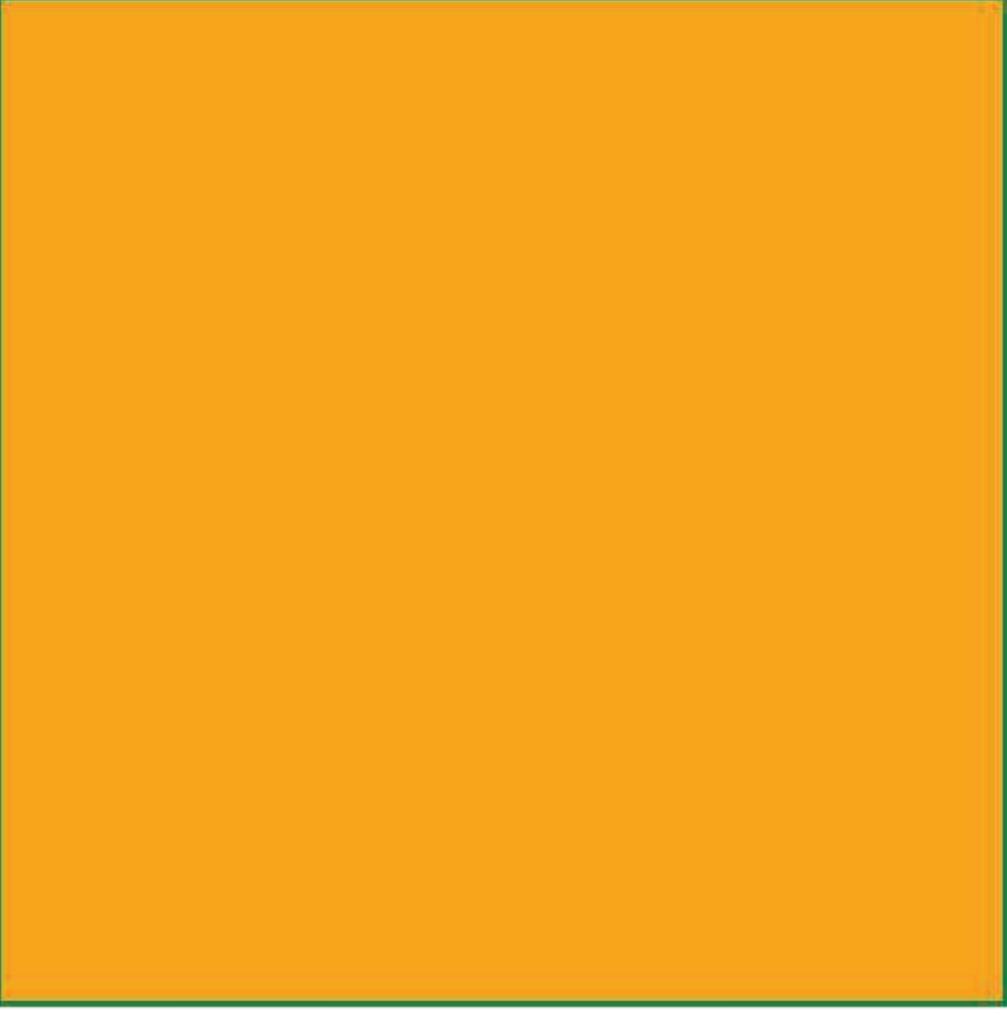
In order to delete an element's child elements, the `empty()` method is used. For instance, consider the following example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("button").click(function() {
                $("#d").empty();
            });
        });
    </script>
</head>
<body>
    <div id="d" style="height:500px;width:500px;border:3px solid green;
background-color:orange;">
        We are using div element in our example.
        <p>A button is used for deletion.</p>
        <p>The button calls out the remove() method to delete the element.</p>
    </div>
    <br>
    <button>Delete the div element from the page.</button>
</body>
</html>
```

This code shows the following result in a browser window with orange background and green border.



After the button is clicked, the following result will be displayed, where the text disappears.



Delete the `div` element from the page.



Can jQuery connect with databases?

5.8 | jQuery with CSS

In addition to HTML, jQuery can also manipulate CSS elements. In this section, we will learn various jQuery methods to add, delete, or modify CSS properties.

5.8.1 `addClass()`

By using `addClass()`, CSS class attributes can be added in various HTML elements. You can pick a single or multiple elements and assign a class with it. For example, in the following example, we have assigned a CSS class “green” with our first heading and paragraph to change its color. Similarly, we have modified the size of our `div` element by using a “size” class.

```

<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
  $("#b1").click(function(){
    $("h1, p").addClass("green");
    $("div").addClass("size");
  });
});
</script>
<style>
.size {
  font-size: xx-small;
}
.green {
  color: green;
}
</style>
</head>
<body>
<h1>The is the first heading!</h1>
<h2>The is the second heading!</h2>
<p>This is the first paragraph.</p>
<p>This is the second paragraph.</p>
<div>This is a piece of vital information.</div><br>
<button id ="b1">Push classes in elements</button>
</body>
</html>

```

This code shows the following result in the browser window.

The is the first heading!

The is the second heading!

This is the first paragraph.

This is the second paragraph.

This is a piece of vital information.

After the button click, the following result will be displayed, which turns the heading and p element content into green color and size of div element.

The is the first heading!

The is the second heading!

This is the first paragraph.

This is the second paragraph.

This is a piece of vital information.

5.8.2 removeClass()

To delete any selected class attribute from multiple elements, you can use the `removeClass()` method. In the following example, we have eliminated the changes of CSS code.

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function(){
            $("button").click(function(){
                $("h1, h2, p").removeClass("green");
            });
        });
    </script>
    <style>
        .green {
            color: green;
        }
    </style>
</head>
<body>
    <h1 class="green">First Green Heading</h1>
    <h2 class="green">Second Green Heading</h2>
    <p class="green">This paragraph is green.</p>
    <p>This paragraph has no colour.</p>
    <button>Delete the CSS class from the selected elements.</button>
</body>
</html>
```

This code shows the following result in a browser window with h1, h2, and first p tag content in green.

First Green Heading

Second Green Heading

This paragraph is green.

This paragraph has no colour.

Delete the CSS class from the selected elements.

After the button click, the following result will be displayed.

First Green Heading

Second Green Heading

This paragraph is green.

This paragraph has no colour.

Delete the CSS class from the selected elements.

5.8.3 toggleClass()

By using the jQuery `toggleClass()` method, we can swap the functionalities of `addClass()` and `removeClass()` methods, that is, it adds and deletes classes from HTML's elements. In the following example, we have done exactly the same.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
  </script>
    <script>
      $(document).ready(function(){
        $("button").click(function(){
          $("h1, p").toggleClass("green");
        });
      });
    </script>
    <style>
      .green{
        color: green;
      }
    </style>
  </head>
  <body>
    <h1>This is the first heading which is soon to be turned green!</h1>
    <h2>This is the second heading which is going to remain the same!</h2>
    <p>This is the first paragraph which is soon to be turned green.</p>
    <p>This is the second paragraph which is soon to be turned green.</p>
    <button>Use the toggleClass function</button>
  </body>
</html>
```

This code shows the following result in the browser window.

This is the first heading which is soon to be turned green!

This is the second heading which is going to remain the same!

This is the first paragraph which is soon to be turned green.

This is the second paragraph which is soon to be turned green.

Use the toggleClass function

5.8.4 css()

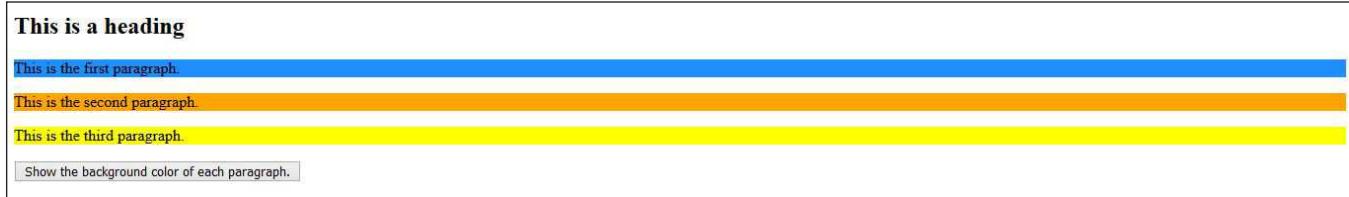
In order to set or return style properties for the specified elements, the `css()` method is used. In our example, we have assigned a separate color to each of our paragraph. An alert is used to display the CSS property of background color for each paragraph which shows the RGB values.

```

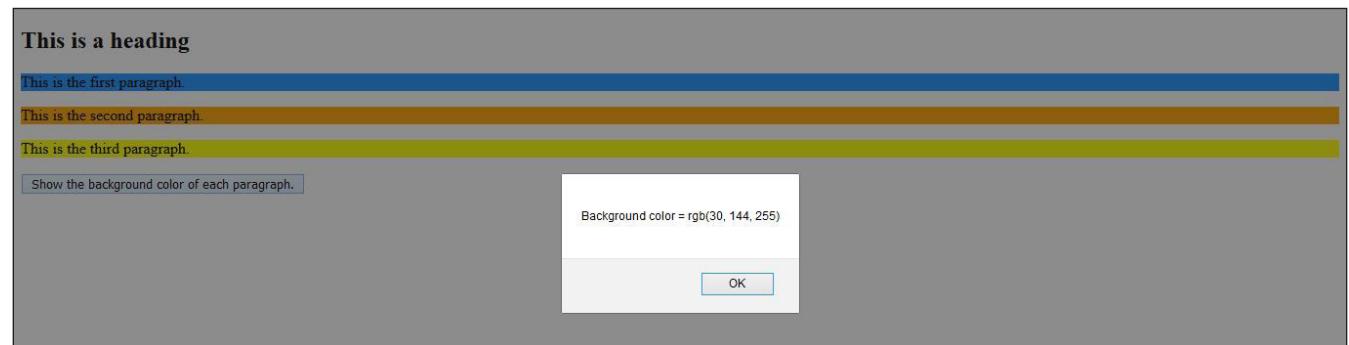
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("button").click(function() {
                alert("Background color = " + $("#p1").css("background-color"));
                alert("Background color = " + $("#p2").css("background-color"));
                alert("Background color = " + $("#p3").css("background-color"));
            });
        });
    </script>
</head>
<body>
    <h2>This is a heading</h2>
    <p id="p1" style="background-color:dodgerblue">This is the first paragraph.</p>
    <p id="p2" style="background-color:orange">This is the second paragraph.</p>
    <p id="p3" style="background-color:yellow">This is the third paragraph.</p>
    <button>Show the background color of each paragraph.</button>
</body>
</html>

```

This code shows the following result in the browser window.



After the button click, the following result will be displayed.



You can also use the CSS method to adjust an already set property in CSS. For instance, we had set colors for our paragraphs. By using the CSS method, we have modified them by accessing and changing the color property.

```

<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function() {
    $("button").click(function() {
        $("#p1").css("background-color", "red");
        $("#p2").css("background-color", "blue");
        $("#p3").css("background-color", "green");
    });
});
</script>
</head>
<body>
<h2>How to set a CSS property? </h2>
<p id="p1" style="background-color:dodgerblue">This is the first paragraph.</p>
<p id="p2" style="background-color:orange">This is the second paragraph.</p>
<p id="p3" style="background-color:yellow">This is the third paragraph.</p>
<p>This paragraph is without any color.</p>
<button>Adjust the background color of the paragraphs.</button>
</body>
</html>

```

This code shows the following result in the browser window.

How to set a CSS property?

This is the first paragraph.

This is the second paragraph.

This is the third paragraph.

This paragraph is without any color.

After the button click, the following result will be displayed.

How to set a CSS property?

This is the first paragraph.

This is the second paragraph.

This is the third paragraph.

This paragraph is without any color.

5.9 | Traversing

jQuery provides excellent traversing functionality. Traversing allows moving around HTML elements in accordance with their link to different elements. By beginning with one selection, you can go further and proceed with your filtering till you get to your intended element.

5.9.1 Ancestors

Ancestor refers to an element which may be a parent or grandparent of a child element. jQuery helps in traversal of the DOM tree by searching any element. There are three methods in jQuery to do this – `parent()`, `parents()`, and `parentsUntil()`.

Figure 5.1 shows an example DOM tree which starts with Document and has a root element named `<html>`. This is followed by two main elements `<head>` and `<body>`. `<head>` contains header related information like page title, meta data,

etc. And <body> contains all the elements which will be displayed on a browser window. Every element has various attributes and most of the elements may also contain text. See the following <button> element example which has attribute “type” and text which can be specified by the developer.

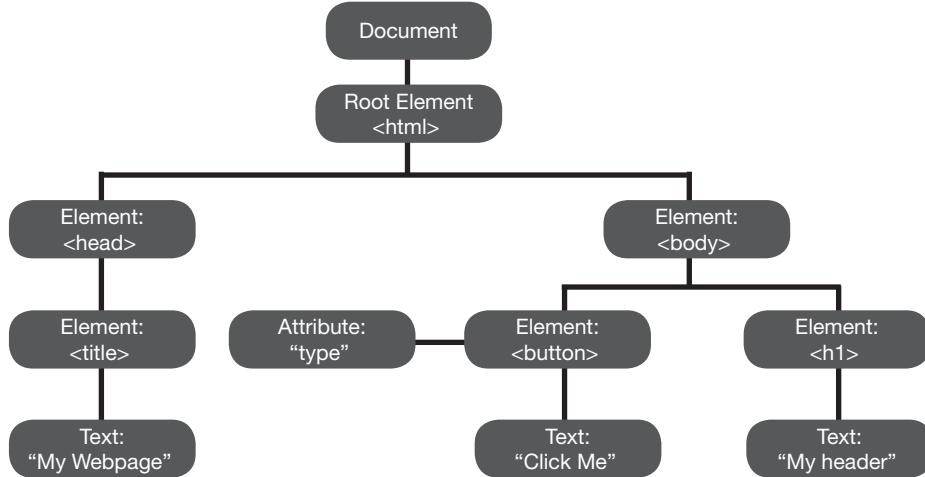


Figure 5.1 DOM tree.

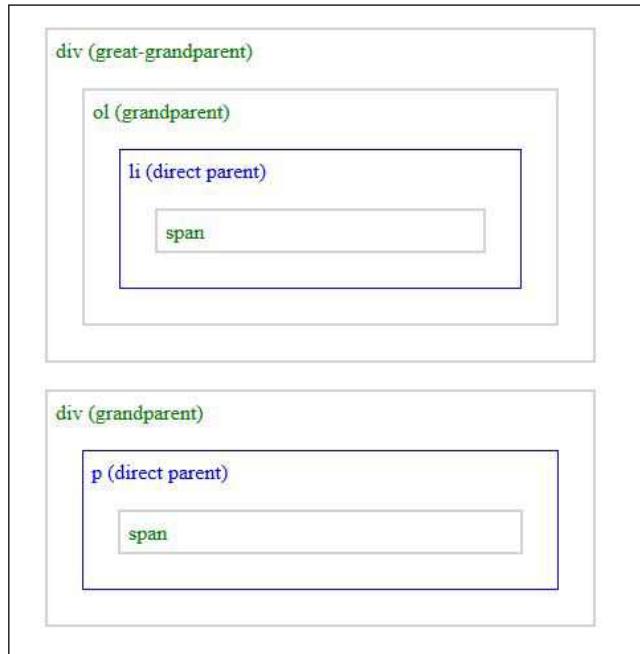
5.9.1.1 parent()

To view the direct parent of a selected element, you can use the `parent()` method. It merely traverses one step up in the DOM. For example, the following code displays the direct parent for all the span elements.

```

<!DOCTYPE html>
<html>
<head>
<style>
.outermost * {
  display: block;
  border: 2px solid lightgrey;
  color: green;
  padding: 6px;
  margin: 20px;}
</style>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("span").parent().css({"color": "blue", "border": "1px solid blue"});
});
</script>
</head>
<body>
<div class="outermost">
<div style="width:400px;">div (great-grandparent)
  <ol>ol (grandparent)
    <li>li (direct parent)
      <span>span</span>
    </li>
  </ol>
</div>
<div style="width:400px;">div (grandparent)
  <p>p (direct parent)
    <span>span</span>
  </p>
</div>
</div>
</body>
</html>
  
```

This code shows the following result in the browser window.



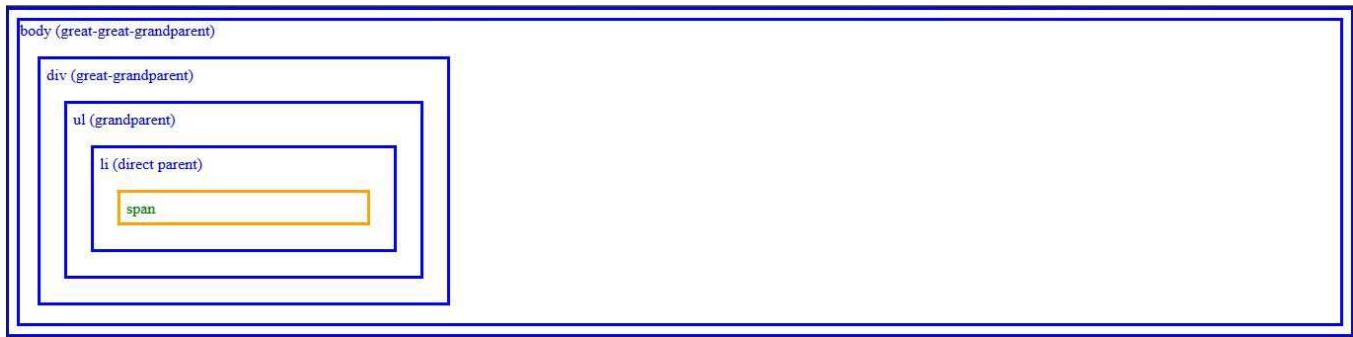
5.9.1.2 parents()

Likewise, the `parents()` method can be used to display all the ancestors from the direct parent to the outermost parent, that is, the root html tag.

```

<!DOCTYPE html>
<html>
<head>
  <style>
    .outermost * {
      display: block;
      border: 3px solid orange;
      color: green;
      padding: 6px;
      margin: 18px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
  <script>
    $(document).ready(function() {
      $("span").parents().css({"color": "blue", "border": "3px solid blue"});
    });
  </script>
</head>
<body class="outermost">body (great-great-grandparent)
  <div style="width:400px;">div (great-grandparent)
    <ul>ul (grandparent)
      <li>li (direct parent)
        <span>span</span>
      </li>
    </ul>
  </div>
</body>
</html>
  
```

This code shows the following result in the browser window.



5.9.1.3 parentsUntil()

To search for all the ancestors till a specific point, you can use the `parentsUntil()` method. For instance, in the following example, you can get all parents starting from the `span` tag and ending at the `div` tag.

```

<!DOCTYPE html>
<html>
<head>
  <style>
    .outermost * {
      display: block;
      border: 2px solid orange;
      color: lightgrey;
      padding: 6px;
      margin: 18px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
  <script>
    $(document).ready(function() {
      $("span").parentsUntil("div").css({"color": "blue", "border": "2px solid blue"});
    });
  </script>
</head>
<body class="outermost"> body (great-great-grandparent)
  <div style="width:400px;">div (great-grandparent)
    <ul>ul (grandparent)
      <li>li (direct parent)
        <span>span</span>
      </li>
    </ul>
  </div>
</body>
</html>
  
```

This code shows the following result in the browser window.



5.9.2 Descendants

As jQuery provides functionality to go up in the DOM tree, it also assists in going down in the DOM tree to select any descendant. A descendant can be any child or grandchild of an element. There are two main methods to find descendants:

1. `children()`
2. `find()`

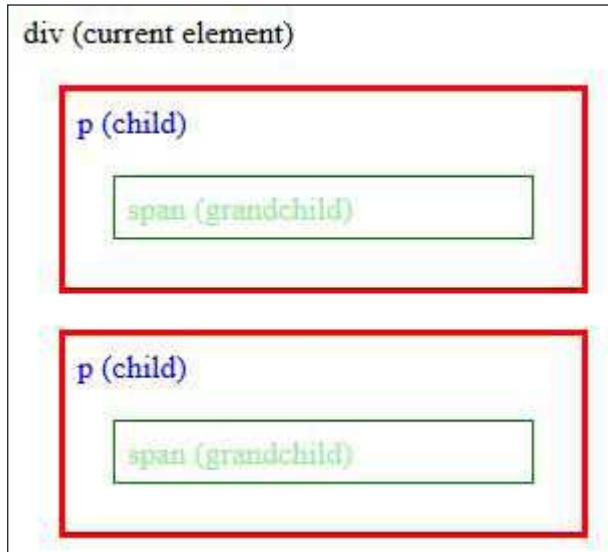
5.9.2.1 `children()`

To view the direct children of an element, you can use the `children()` method. It merely goes down to show a single level in the DOM. For instance, to check the direct child of the `div` tag, we have used the `children()` method by a blue-color representation.

```

<!DOCTYPE html>
<html>
<head>
  <style>
    .innermost * {
      display: block;
      border: 1px solid green;
      color: lightgreen;
      padding: 6px;
      margin: 18px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
  <script>
$(document).ready(function(){
  $("div").children().css({"color": "blue", "border": "3px solid red"});
});
  </script>
</head>
<body>
  <div class="innermost" style="width:300px;">div (current element)
    <p>p (child)
      <span>span (grandchild)</span>
    </p>
    <p>p (child)
      <span>span (grandchild)</span>
    </p>
  </div>
</body>
</html>
  
```

This code shows the following result in the browser window.



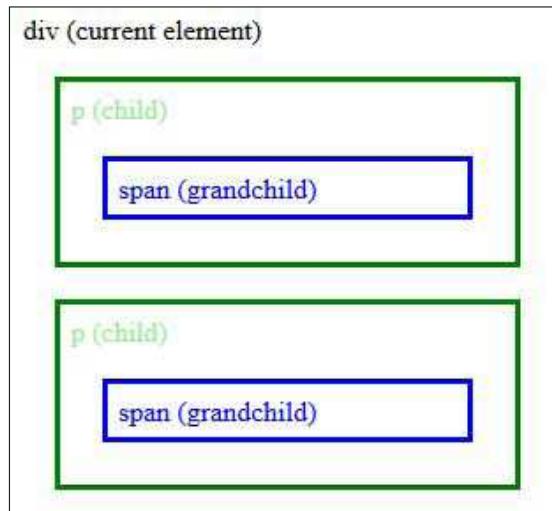
AQ1

5.9.2.2 find()

To view all the existing children of a specified element, the `find()` method is used. Consider the following example in which we will find all the span element which are child to div element and css color properties.

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .innermost * {
      display: block;
      border: 3px solid green;
      color: lightgreen;
      padding: 6px;
      margin: 18px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
  <script>
    $(document).ready(function() {
      $("div").find("span").css({"color": "blue", "border": "3px solid blue"});
    });
  </script>
</head>
<body>
  <div class="innermost" style="width:300px;">div (current element)
    <p>p (child)
      <span>span (grandchild)</span>
    </p>
    <p>p (child)
      <span>span (grandchild)</span>
    </p>
  </div>
</body>
</html>
```

This code shows the following result in the browser window.



5.9.3 Siblings

Siblings elements are the ones who share the same parent. jQuery allows to traverse horizontally to search an element's siblings. To check a sibling, there are multiple methods such as `siblings()`, `next()`, `nextAll()`, `nextUntil()`, `prev()`, `prevAll()`, and `prevUntil()`.

5.9.3.1 `siblings()`

To check the siblings of a specified element, you can use the `siblings()` method. For example, to check the siblings of the second heading `<h2>`, we have used the `siblings()` method.

```

<!DOCTYPE html>
<html>
<head>
    <style>
        .s * {
            display: block;
            border: 3px solid yellow;
            color: orange;
            padding: 6px;
            margin: 18px;
        }
    </style>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script>
        $(document).ready(function() {
            $("h2").siblings().css({"color": "blue", "border": "3px solid blue"});
        });
    </script>
</head>
<body class="s">
    <div>div (parent)
        <p>p</p>
        <span>span</span>
        <h2>h2</h2>
        <h3>h3</h3>
        <p>p</p>
    </div>
</body>
</html>
  
```

This code shows the following result in the browser window.



5.9.3.2 next()

To view the next sibling of a specified element, you can use the `next()` method. For example, to check the next sibling of `h2` and `h3`, we have written the following code:

```
<!DOCTYPE html>
<html>
<head>
<style>
.s * {
display: block;
border: 3px solid green;
color: green;
padding: 6px;
margin: 18px;
}
</style>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("h2").next().css({"color": "blue", "border": "3px solid blue"});
});
$(document).ready(function(){
    $("h3").next().css({"color": "blue", "border": "3px solid blue"});
});
</script>
</head>
<body class="s">
<div>div (parent)
<p>p</p>
<span>span</span>
<h2>h2</h2>
<h3>h3</h3>
<p>p</p>
</div>
</body>
</html>
```

This code shows the following result in the browser window.



5.9.3.3 nextAll()

To view all the siblings which are next to a selected element, you can use the `nextAll()` method. For example, to view h2's next siblings you can write the following code:

```

<!DOCTYPE html>
<html>
<head>
  <style>
    .s * {
      display: block;
      border: 3px solid green;
      color: green;
      padding: 6px;
      margin: 18px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
  <script>
    $(document).ready(function(){
      $("h2").nextAll().css({"color": "blue", "border": "3px solid blue"});
    });
  </script>
</head>
<body class="s">
  <div>div (parent)
    <p>p1</p>
    <p>p2</p>
    <span>span</span>
    <h2>h2</h2>
    <h3>h3</h3>
    <span>span</span>
    <p>p3</p>
  </div>
</body>
</html>
  
```

This code shows the following result in the browser window.



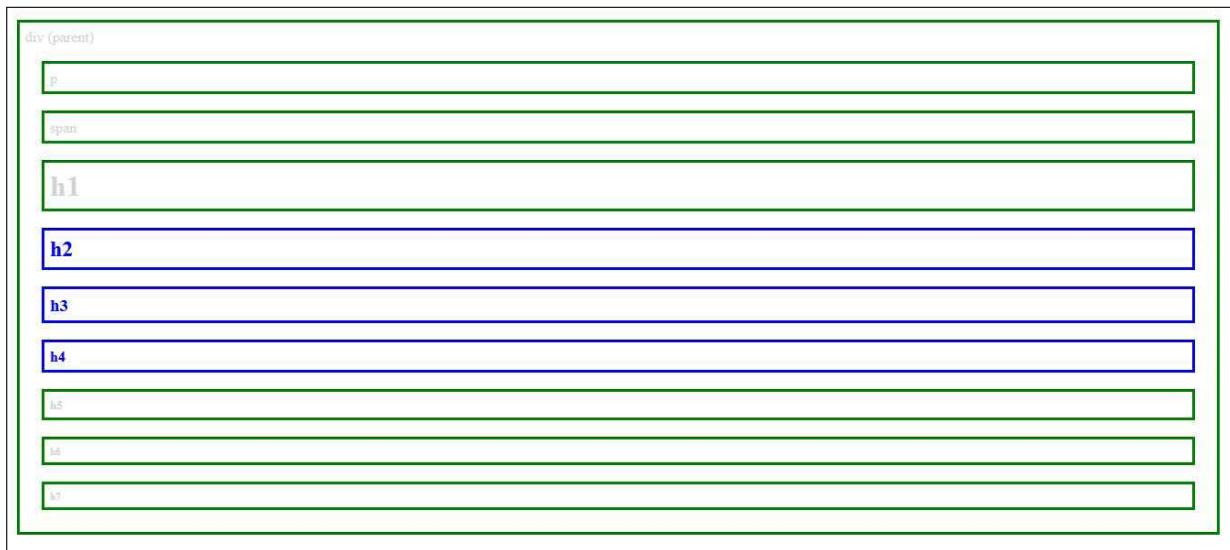
5.9.3.4 nextUntil()

To view the next siblings of element till a given point is reached, you can use the `nextUntil()` method. In our example, we started searching for the next siblings of `h1` till `h5`.

```

<html>
<head>
  <style>
    .s * {
      display: block;
      border: 3px solid green;
      color: lightgrey;
      padding: 6px;
      margin: 18px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
  <script>
    $(document).ready(function() {
      $("h1").nextUntil("h5").css({"color": "blue", "border": "3px solid blue"});
    });
  </script>
</head>
<body class="s">
  <div>div (parent)
    <p>p</p>
    <span>span</span>
    <h1>h1</h1>
    <h2>h2</h2>
    <h3>h3</h3>
    <h4>h4</h4>
    <h5>h5</h5>
    <h6>h6</h6>
    <h6>h7</h6>
  </div>
</body>
</html>
    
```

This code shows the following result in the browser window.



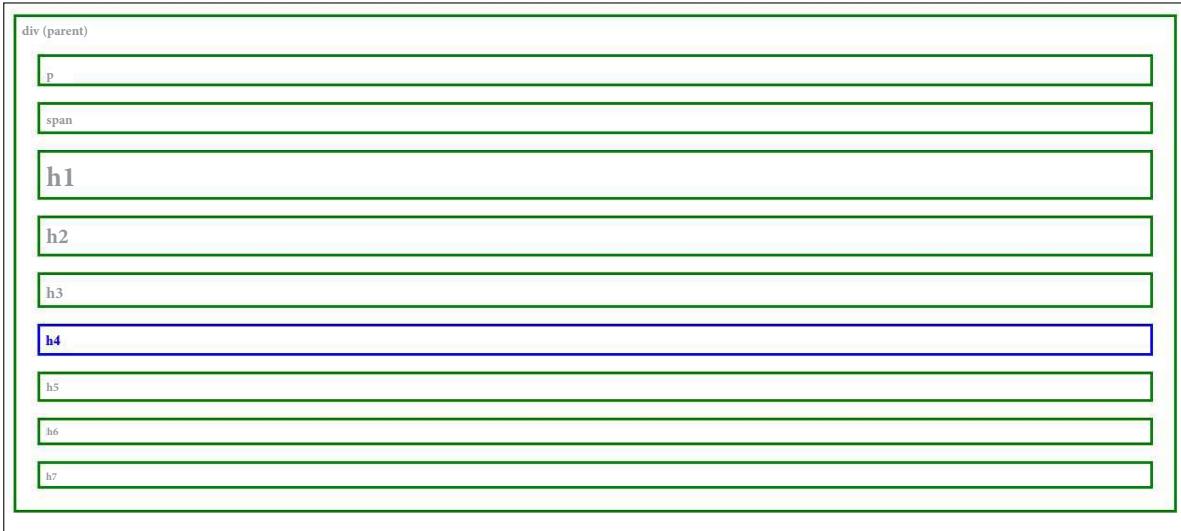
5.9.3.5 prev()

You can also check the previous elements of a sibling by using the `prev()` method. Consider the following example where we check the immediate previous sibling of `h5`:

```

<!DOCTYPE html>
<html>
<head>
    <style>
        .s * {
            display: block;
            border: 3px solid green;
            color: lightgrey;
            padding: 6px;
            margin: 18px;
        }
    </style>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script>
        $(document).ready(function(){
            $("h5").prev().css({"color": "blue", "border": "3px solid blue"});
        });
    </script>
</head>
<body class="s">
    <div>div (parent)
        <p>p</p>
        <span>span</span>
        <h1>h1</h1>
        <h2>h2</h2>
        <h3>h3</h3>
        <h4>h4</h4>
        <h5>h5</h5>
        <h6>h6</h6>
        <h6>h7</h6>
    </div>
</body>
</html>
  
```

This code shows the following result in the browser window.

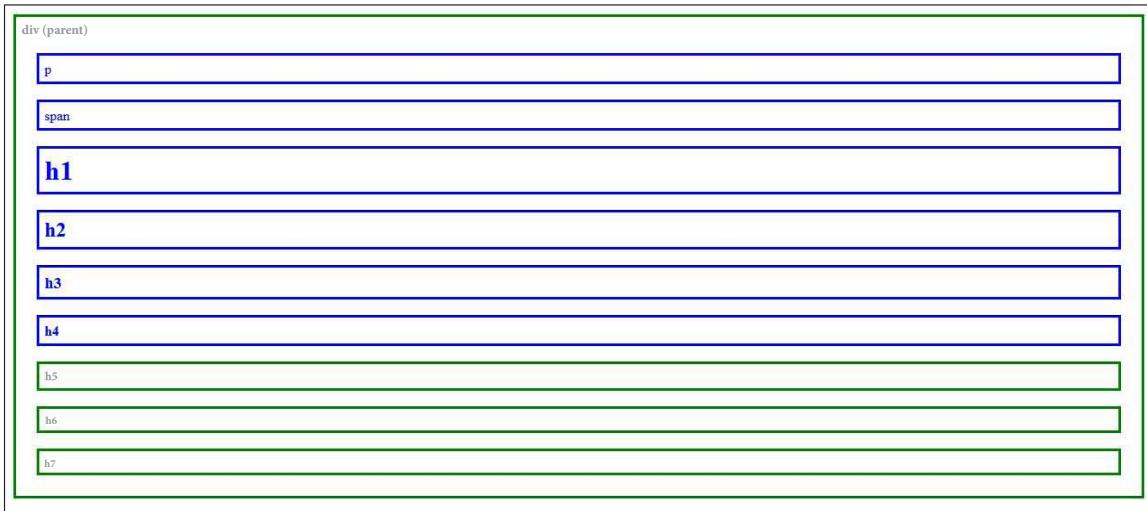


5.9.3.6 prevAll()

Similarly, to check all the previous siblings of *h5*, you can use the `prevAll()` method.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .s * {
            display: block;
            border: 3px solid green;
            color: lightgrey;
            padding: 6px;
            margin: 18px;
        }
    </style>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script>
        $(document).ready(function() {
            $("h5").prevAll().css({"color": "blue", "border": "3px solid blue"});
        });
    </script>
</head>
<body class="s">
    <div>div (parent)
        <p>p</p>
        <span>span</span>
        <h1>h1</h1>
        <h2>h2</h2>
        <h3>h3</h3>
        <h4>h4</h4>
        <h5>h5</h5>
        <h6>h6</h6>
        <h6>h7</h6>
    </div>
</body>
</html>
```

This code shows the following result in the browser window.



5.9.3.7 prevUntil()

Additionally, to check the previous siblings of *h5* to a certain extent you can use the `prevUntil()` method.

```

<!DOCTYPE html>
<html>
<head>
  <style>
    .s * {
      display: block;
      border: 3px solid green;
      color: lightgrey;
      padding: 6px;
      margin: 18px;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
  </script>
  <script>
    $(document).ready(function(){
      $("h5").prevUntil("span").css({"color": "blue", "border": "3px solid blue"});
    });
  </script>
</head>
<body class="s">
  <div>div (parent)
    <p>p</p>
    <span>span</span>
    <h1>h1</h1>
    <h2>h2</h2>
    <h3>h3</h3>
    <h4>h4</h4>
    <h5>h5</h5>
    <h6>h6</h6>
    <h6>h7</h6>
  </div>
</body>
</html>
  
```

This code shows the following result in the browser window.



5.9.4 Filtering

You can also traverse through filtering.

5.9.4.1 `first()`

To view the beginning element of a defined element, you can use the `first()` method. In this example, we have selected the beginning `<div>` element.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("div").first().css("background-color", "lightgreen");
});
</script>
</head>
<body>
<h1>How To Use First Method for Traversing?</h1>
<p>This paragraph does not belong to any div element.</p>
<div style="border: 3px solid blue;">
    <p>This is the first paragraph in this div.</p>
    <p>This is the second paragraph in this div.</p>
</div>
<br>
<div style="border: 3px solid blue;">
    <p>This is the first paragraph in this div.</p>
    <p>This is the second paragraph in this div.</p>
</div>
<br>
<div style="border: 3px solid blue;">
    <p>This is the first paragraph in this div.</p>
    <p>This is the second paragraph in this div.</p>
</div>
</body>
</html>
```

This code shows the following result in the browser window.

How To Use First Method for Traversing?

This paragraph does not belong to any div element.

This is the first paragraph in this div.

This is the second paragraph in this div.

This is the first paragraph in this div.

This is the second paragraph in this div.

This is the first paragraph in this div.

This is the second paragraph in this div.

5.9.4.2 last()

Similarly, by using the `last()` method you can check the ending element of the selected elements. For example,

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("div").last().css("background-color", "lightgreen");
        });
    </script>
</head>
<body>
    <h1>How To Use First Method for Traversing?</h1>
    <p>This paragraph does not belong to any div element.</p>
    <div style="border: 3px solid blue;">
        <p>This is the first paragraph in this div.</p>
        <p>This is the second paragraph in this div.</p>
    </div>
    <br>
    <div style="border: 3px solid blue;">
        <p>This is the first paragraph in this div.</p>
        <p>This is the second paragraph in this div.</p>
    </div>
    <br>
    <div style="border: 3px solid blue;">
        <p>This is the first paragraph in this div.</p>
        <p>This is the second paragraph in this div.</p>
    </div>
</body>
</html>
```

This code shows the following result in the browser window.

How To Use First Method for Traversing?

This paragraph does not belong to any div element.

This is the first paragraph in this div.

This is the second paragraph in this div.

This is the first paragraph in this div.

This is the second paragraph in this div.

This is the first paragraph in this div.

This is the second paragraph in this div.

5.9.4.3 eq()

To select a specific element, you can use the `eq()` method. It takes the index number as an argument for the defined elements. The index begins with 0. Consider the following example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
    <script>
        $(document).ready(function() {
            $("div").eq(1).css("background-color", "lightgreen");
        });
    </script>
</head>
<body>
    <h1>How To Use First Method for Traversing?</h1>
    <p>This paragraph does not belong to any div element.</p>
    <div style="border: 3px solid blue;">
        <p>This is the first paragraph in this div.</p>
        <p>This is the second paragraph in this div.</p>
    </div>
    <br>
    <div style="border: 3px solid blue;">
        <p>This is the first paragraph in this div.</p>
        <p>This is the second paragraph in this div.</p>
    </div>
    <br>
    <div style="border: 3px solid blue;">
        <p>This is the first paragraph in this div.</p>
        <p>This is the second paragraph in this div.</p>
    </div>
</body>
</html>
```

This code shows the following result in the browser window.

How To Use First Method for Traversing?

This paragraph does not belong to any div element.

This is the first paragraph in this div.

This is the second paragraph in this div.

This is the first paragraph in this div.

This is the second paragraph in this div.

This is the first paragraph in this div.

This is the second paragraph in this div.



Can you extend current jQuery functionality? Hint: Creating a plugin.

Summary

jQuery is one of the popular frameworks for web application development. It greatly simplifies the use of JavaScript and provides various functionalities that ease the front-end development. Since the popularity of jQuery has increased significantly, there are a lot of plugins available which enhance the front-end processing capabilities.

In this chapter, we have learned the following concepts:

1. jQuery and its use to control HTML.
2. Various uses of jQuery for HTML tags.
3. jQuery selectors to manipulate content within elements.
4. Access parent elements and traverse between DOM elements to find first, last, etc.

In Chapter 6, we will learn about Bootstrap and how it is useful to develop a mobile-friendly application. It offers functionalities that help to design a slick and modern webpage, which will render properly on all the available screen sizes.

Multiple-Choice Questions

1. Which one of the following jQuery methods prevents the browser from executing the default action?
 - (a) StopImmediatePropagation()
 - (b) StopPropagation()
 - (c) preventDefault()
 - (d) None of the above
 2. We can pass an anonymous function as an argument to another function.
 - (a) True
 - (b) False
 3. Which one of the built-in methods in JavaScript sorts the elements of an array?
 - (a) Sort()
 - (b) Order()
- (c) changeOrder(order)
 - (d) None of the above
 4. Which of the given jQuery methods removes all child nodes from the set of matched elements?
 - (a) delete()
 - (b) empty()
 - (c) remove(expr)
 - (d) None of the above
 5. Which one of the given jQuery methods filters out elements from a set of matched elements?
 - (a) setFilter(selector)
 - (b) filter(selector)
 - (c) putFilter(selector)
 - (d) None of the above

Review Questions

1. How can you select a particular tag to perform an action on it?
2. How can you select multiple tags at once to perform a common action on them?
3. How can you remove an HTML element on a button click?
4. How can you find the last element of a particular type of element like <div>?
5. How can you add style to the first element of a particular type of element like <a>?
6. How can you find parent of an element in nested elements case?
7. How do you add jQuery library to a page?
8. Can you change class of an element on a button click?
9. Can you add a class to an element on a button click?
10. How can you read value from a text box using jQuery?
11. Can you use jQuery and JavaScript code together on a page?
12. Is jQuery operating system dependent?
13. Can you have nested “document.ready” functions?
14. Do you need to install anything on the server to add jQuery code?

Exercises

1. Create a page which contains a table and add button which will add rows to this table.
2. Change color of a text on button click.
3. Show a popup box without using JavaScript alert.
4. Write code to fetch data from Wikipedia by using ajax function.
5. Add a text box and a button. Write code to remove all the elements which ID matches with the text entered in the text box upon clicking on the button.

Project Idea

Take the dictionary project idea from the previous chapters. Create a page that allows adding a word to the word list. This page should provide a text box and allow users to enter a word that they want to add to the dictionary list. It should

also have 3 text areas where they can add example sentences, synonyms, and antonyms of the given word. In the end, add a button that will add the word and its example, synonyms, and antonyms to the word list table upon clicking on it.

Recommended Readings

1. Bear Bibeault, Yehuda Katz, Aurelio De Rosa, Dave Methvin, John Resig. 2016. *jQuery In Action, Third Edition, (MANNING)*. Dreamtech Press: New Delhi
2. Jon Raasch, Graham Murray, Vadim Ogievetsky, Joseph Lowery. 2015. *Javascript and jQuery for Data Analysis and Visualization*. Wiley: New Jersey
3. Richard York. 2015. *Web Development with jQuery*. Wrox: Birmingham
4. jQuery – <https://jquery.com/>
5. W3School – <https://www.w3schools.com/jquery/>
6. jQuery UI – <https://jqueryui.com/>

Introduction to Bootstrap

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- What is Bootstrap.
- How to setup Bootstrap library.
- Structure of a Bootstrap-enabled webpage.
- Various Bootstrap elements like grid, colors, images, etc.
- How to make your application responsive.

6.1 | Overview of Bootstrap



Bootstrap is one of the most prominent and leading front-end frameworks. Bootstrap brings out a whole new set of designs and functionalities in HTML, CSS, and even JavaScript components. Bootstrap can change anything on the webpage – buttons, text, tables, links, images, and other elements – while it introduces some of its built-in features which decrease the need of writing HTML and CSS from scratch.

Bootstrap was introduced in the beginning of 2010 when Jacob Thornton and Mark Otto from Twitter released Bootstrap as an open-source project. After three years, Bootstrap's success led it to become the top project on GitHub.

Most importantly, Bootstrap supported “responsiveness”: a contemporary website approach known as *responsive web design*. Traditionally, websites were created with the knowledge that users would open and interact with them on desktop PCs. Therefore, the imagination and creativity of website designers was only restricted to desktop PCs. However, the transformation of phones into “smartphones” in the second half of 2000s changed the web scene completely. With smartphones, tablets, smart watches, and other range of computing devices along with the emergence of Internet of Things, web designers had to improvise. The websites which were designed for desktops were simply not compatible on smaller screens. Therefore, user experience was affected badly.

Here, responsive design came up as a workable solution through which websites could automatically recalibrate their structure according to the device of the user. Bootstrap was one of the earlier tools to support this responsiveness and thus ultimately the word “responsiveness” became synonymous with Bootstrap.



What is the difference between jQuery and Bootstrap?

6.1.1 Prerequisites

Before beginning the chapter, it is important to note that Bootstrap requires a basic grasp of HTML and CSS. If you are unfamiliar with HTML and CSS, then please check our previous chapters on HTML and CSS.

6.1.2 Installation

In this chapter, we are going to use Bootstrap 4. You can either download it from <https://getbootstrap.com> or add it in your website via Content Delivery Network (CDN). In our chapter, we would adopt the latter approach from the MaxCDN. This strategy is applied because it provides quicker site speed.

**QUICK
CHALLENGE**

Review other CDN platforms and come up with a comparison chart that shows various features and functionalities.

6.2 | Structure of a Bootstrap-enabled Webpage

In this section, we will explore the structure of an HTML webpage to make it suitable for Bootstrap. Consider the following example to generate a basic Bootstrap-powered webpage.

1. In the beginning you must place the HTML5 doctype, the language attribute, and the appropriate character set as follows:

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-7">
  </head>
</html>
```

2. As we already know, Bootstrap was originally designed for smartphones. Therefore, it is common to find mobile-first styles in it. For incorporating the appropriate touch zooming and rendering, make sure to go inside the `<head>` element and add a `<meta>` tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

When the page is initially loaded in a browser, the “`initial-scale=1`” sets the beginning zoom level. For setting the page’s width to correspond it with the device’s screen width, the “`width=device-width`” is used.

3. In Bootstrap, the contents of the website are encompassed in an element known as a container. Container has two classes:
 - `.container`
 - `.container-fluid`

To understand the difference between both classes, check the following two diagrams to determine their positioning of width.



.container class



.container-fluid

As you can see in the above pictures, `.container` class has fixed width which is applicable for all the screen sizes (xs, sm, md, and lg). Following are the details of the screen sizes:

- **xs:** Extra small devices (less than 768; e.g., smart phones).
- **sm:** Small screens (from 768 pixels and up; e.g., tablets).
- **md:** Medium screen (>= 992 pixels; e.g., desktops or laptops).
- **lg:** Large screens (>= 1200 pixels; e.g., large desktops).

On the other hand, `.container-fluid` expands to the fullest width of the page’s viewport. Hence, depending on the available viewport of the page, `.container` gives a specific width to the div tag. However, this line does not exist for `.container-fluid` as it changes according to the viewport width.



What are the two ways you can display Bootstrap code?

6.2.1 Container Example

Let us go through an example to understand the container element:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Container Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>

<div class="container">
    <h1>This heading is placed in a container class.</h1>
    <p>This paragraph is placed in a container class.</p>
    <p>This paragraph is placed in a container class.</p>
</div>
</body>
</html>
```

As you execute the code, you would realize that output is exactly the same as an HTML webpage. So, what exactly did the `.container` class do since HTML's `div` element can apparently serve the same container property? Well, actually the container class is not a simple container; it provides responsiveness for the width of its contents. In the upcoming examples, you would begin to see a clearer picture and impact of Bootstrap on webpages.



What effect will Bootstrap have on page if we donot use container?

6.3 | Grids



Grid systems are one of the most prominent features of Bootstrap. Grid is created through a flexbox and can accept up to 12 columns in a webpage. The lesser the number of columns, the wider the columns are. Due to Bootstrap's responsiveness, this grid system rearranges the columns according to the screen size.



What is a Bootstrap container?

6.3.1 Classes

There are five types of classes in grids:

1. **.col:** This is used for smallest screens where the width of the screen falls under 576px.
2. **.col-sm:** This is used for small screens where the width of the screen exceeds 576px.
3. **.col-md:** This is used for medium screens where the width of the screen exceeds 768px.
4. **.col-lg:** This is used for large screens where the width of the screen exceeds 992px.
5. **.col-xl:** This is used for extra-large screens where the width of the screen exceeds 1200px.

To generate four columns of equal size, consider the following example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Columns Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container-fluid">
    <h1>Four Columns</h1>
    <div class="row">
        <div class="col" style="background-color:red; border-style: dotted;">.col</div>
        <div class="col" style="background-color:red; border-style: dotted">.col</div>
        <div class="col" style="background-color:red; border-style: dotted">.col</div>
        <div class="col" style="background-color:red; border-style: dotted">.col</div>
    </div>
</div>
</body>
</html>
```

The above code produces the following output. In this, the following section of the code defines the grid we want to create. Outer div element start with “row” class and inner div starts with “col” class.

```
<div class="row">
    <div class="col" style="background-color:red; border-style: dotted;">.col</div>
    <div class="col" style="background-color:red; border-style: dotted">.col</div>
    <div class="col" style="background-color:red; border-style: dotted">.col</div>
    <div class="col" style="background-color:red; border-style: dotted">.col</div>
</div>
```

Four Columns



For practice, add more columns by repeating the *div* tag with “*col*” and you can see automatic readjustments of columns.

Now, check the following example in which we have generated four columns of equal widths. This column would display properly in PCs and tablets but for devices with less than 576px, the columns would readjust and sit on top of each other, like a stack of books. Try resizing the window and make it smaller to view the stack.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Bootstrap Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container-fluid">
    <h1>Column Example</h1>
    <p>Resize the browser window to see the effect.</p>
    <p>The columns will automatically stack on top of each other when the screen is less than 576px wide.</p>
    <div class="row">
        <div class="col-sm-3" style="background-color:green;">First Column</div>
        <div class="col-sm-3" style="background-color:lightgreen;">Second Column</div>
        <div class="col-sm-3" style="background-color:green;">Third Column</div>
        <div class="col-sm-3" style="background-color:lightgreen;">Fourth Column</div>
    </div>
</div>
</body>
</html>

```

Let us see the above outputs which shows how the grid layout changes according to screen sizes.

On Laptop:

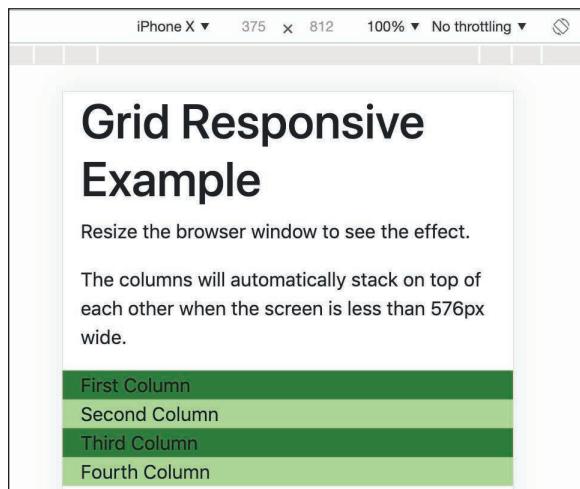
Grid Responsive Example

Resize the browser window to see the effect.

The columns will automatically stack on top of each other when the screen is less than 576px wide.



On iPhone:





What are the benefits of using grid as compared to table?

6.4 | Typography

Bootstrap supports various properties to format text properly for various device sizes. Some of the texts are rendered properly with default behavior, which means without even using any specific classes. By default, Bootstrap 4 assigns the default settings for text-related content (Table 6.1).

Table 6.1 Default settings for text-related content

Property	Default
<i>font-size</i>	16px
<i>line-height</i>	1.5
<i>font-family</i>	Helvetica Neue, Helvetica, Arial, Sans-serif
<i>margin-top for <p></i>	0
<i>margin-bottom for <p></i>	16px
<i>h1</i>	40px
<i>h2</i>	32px
<i>h3</i>	28px
<i>h4</i>	24px
<i>h5</i>	20px
<i>h6</i>	16px

Additionally, headings are bolder in Bootstrap 4.



Can you extend Bootstrap functionality with plugins like jQuery?

6.4.1 Display Headings

Bootstrap also offers another set of headings known as display headings. These headings are more prominent than HTML's headings. In total, there are four types of display headings, each distinguished by their number. They are distinguished due to their font-size and font-weight.

1. *.display-1*
2. *.display-2*
3. *.display-3*
4. *.display-4*

Let us execute the following display heading code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Display Headings</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>

<body>
<div class="container-fluid">
    <h1>This Is Our First Heading</h1>
    <h1 class="display-1">This Is Our First Display Heading!</h1>
    <h1 class="display-2">This Is Our Second Display Heading!</h1>
    <h1 class="display-3">This Is Our Third Display Heading!</h1>
    <h1 class="display-4">This Is Our Fourth Display Heading!</h1>
</div>
</body>
</html>
```

The above code produces the following result which shows various types of headings.

This Is Our First Heading

This Is Our First Display Heading!

This Is Our Second Display Heading!

This Is Our Third Display Heading!

This Is Our Fourth Display Heading!

6.4.2 HTML Text Elements in Bootstrap

Many of HTML's elements offer a different result in Bootstrap. Consider the impact of following properties in Bootstrap:

1. The `<small>` element from HTML is used in Bootstrap 4 for a different purpose. Bootstrap uses it to generate a secondary text which displays a lighter tone.
2. The `<mark>` element is used for highlighting any piece of information with a little padding and a yellow color for its background.
3. When an abbreviation is written via `<abbr>` then Bootstrap 4 adds a dotted border at its bottom.
4. Instead of `<blockquote>` element in HTML, Bootstrap provides the “`.blockquote`” class for quotations.

Consider the following example which implements all these features in a webpage.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>HTML Text Elements</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>HTML Text Elements in Bootstrap</h1>
    <h1>h1 heading <small>Some <mark>information</mark></small></h1>
    <h2>h2 heading <small>Some information</small></h2>
    <h3>h3 heading <small>Some information</small></h3>
    <h4>h4 heading <small>Some information</small></h4>
    <h5>h5 heading <small>Some information</small></h5>
    <h6>h6 heading <small>Some information</small></h6>
    <p>The <abbr title="European Union">EU</abbr> has introduced the GDPR</p>
    <blockquote class="blockquote">
        <p>Europe's new data protection rules will be a reality tomorrow. Europeans' privacy will be better protected and companies benefit from a single set of rules across the EU. Strong data protection rules are the basis for a functioning Digital Single Market and for the online economy to prosper. The new rules ensure that citizens can trust in how their data is used and that the EU can make best of the opportunities of the data economy.
        Our new data protection rules were agreed for a reason: Two-thirds of Europeans are concerned about the way their data was being handled, feeling they have no control over information they give online. Companies need clarity to be able to safely extend operations across the EU. Recent data scandals confirmed that with stricter and clearer data protection rules we are doing the right thing in Europe.</p>
        <footer class="blockquote-footer">Andrus Ansip</footer>
    </blockquote>
</div>
</body>
</html>
```

The above code produces the following result, which shows various types of HTML elements in Bootstrap.

HTML Text Elements in Bootstrap

h1 heading Some information

h2 heading Some information

h3 heading Some information

h4 heading Some information

h5 heading Some information

h6 heading Some information

The EU has introduced the GDPR

Europe's new data protection rules will be a reality tomorrow. Europeans' privacy will be better protected and companies benefit from a single set of rules across the EU. Strong data protection rules are the basis for a functioning Digital Single Market and for the online economy to prosper. The new rules ensure that citizens can trust in how their data is used and that the EU can make best of the opportunities of the data economy. Our new data protection rules were agreed for a reason: Two-thirds of Europeans are concerned about the way their data was being handled, feeling they have no control over information they give online. Companies need clarity to be able to safely extend operations across the EU. Recent data scandals confirmed that with stricter and clearer data protection rules we are doing the right thing in Europe.

— Andrus Ansip

6.5 | Colors

Bootstrap provides several color classes for both the colors of font and their backgrounds.

6.5.1 Color Classes to Convey Meaning

Bootstrap offers quite a few classes which are used to show different colors to define different events. For example, to convey a warning, the `.text-danger` class is used. Table 6.2 explains all the color classes.

Table 6.2 The different color classes

Code	Color
<code>.text-primary</code>	This class shows the text color blue and specifies something significant.
<code>.text-secondary</code>	This class shows the text color grey and specifies something less significant.
<code>.text-success</code>	This class shows the text color green and shows some the sign of success.
<code>.text-danger</code>	This class shows the text color and specifies some danger.
<code>.text-warning</code>	This class shows the text color yellow/orange and specifies warning.
<code>.text-info</code>	This class shows the text color light blue and specifies information.
<code>.text-light</code>	This class shows the light grey text color.
<code>.text-dark</code>	This class shows the dark grey text color.
<code>.text-muted</code>	This class shows the grey text color.
<code>.text-white</code>	This class shows the white text color.

To understand the use of all these color classes, check the following example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Color Classes</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Color Classes</h2>
    <p>Whenever, you want to convey a specific meaning, use the the following classes:</p>
    <p class="text-muted">If text-muted is used, then the text looks like this line.</p>
    <p class="text-primary">If text-primary is used, then the text looks like this line.</p>
    <p class="text-success">If text-success is used, then the text looks like this line.</p>
    <p class="text-info">If text-info is used, then the text looks like this line.</p>
    <p class="text-warning">If text-warning is used, then the text looks like this line.</p>
    <p class="text-danger">If text-danger is used, then the text looks like this line.</p>
    <p class="text-secondary">If text-secondary is used, then the text looks like this line.</p>
    <p class="text-dark">If text-dark is used, then the text looks like this line.</p>
    <p class="text-body">If text-body is used, then the text looks like this line.</p>
    <p class="text-light">If text-light is used, then the text looks like this line.</p>
    <p class="text-white">If text-white is used, then the text looks like this line.</p>
</div>
</body>
</html>
```

The above code produces the following result, which shows various colors for the text.

Color Classes

Whenever, you want to convey a specific meaning, use the the following classes:

If text-muted is used, then the text looks like this line.

If text-primary is used, then the text looks like this line.

If text-success is used, then the text looks like this line.

If text-info is used, then the text looks like this line.

If text-warning is used, then the text looks like this line.

If text-danger is used, then the text looks like this line.

If text-secondary is used, then the text looks like this line.

If text-dark is used, then the text looks like this line.

If text-body is used, then the text looks like this line.

6.5.2 Background Colors

The background color of text can also be configured with Bootstrap's color classes. To check their effect, run the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Bootstrap Web Page</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Contextual Backgrounds</h2>
    <p class="bg-primary text-black">If you use the bg-primary class, then your text would look like this.</p>
    <p class="bg-success text-black">If you use the bg-success class, then your text would look like this.</p>
    <p class="bg-info text-black">If you use the bg-info class, then your text would look like this.</p>
    <p class="bg-warning text-black">If you use the bg-warning class, then your text would look like this.</p>
    <p class="bg-danger text-black">If you use the bg-danger class, then your text would look like this.</p>
    <p class="bg-secondary text-black">If you use the bg-secondary class, then your text would look like this.</p>
    <p class="bg-dark text-white">If you use the bg-dark class, then your text would look like this.</p>
    <p class="bg-light text-black">If you use the bg-light class, then your text would look like this.</p>
</div>
</body>
</html>
```

The above code produces the following result, which shows various background colors.

Contextual Backgrounds

If you use the `bg-primary` class, then your text would look like this.

If you use the `bg-success` class, then your text would look like this.

If you use the `bg-info` class, then your text would look like this.

If you use the `bg-warning` class, then your text would look like this.

If you use the `bg-danger` class, then your text would look like this.

If you use the `bg-secondary` class, then your text would look like this.

If you use the `bg-dark` class, then your text would look like this.

If you use the `bg-light` class, then your text would look like this.

6.6 | Images

Bootstrap also facilitates web designers to display their images in different styles. We will learn the following types of images in this section:

1. Rounded images
2. Circle images
3. Thumbnail images

6.6.1 Rounded Images

To display an image with rounder corners, the `.rounded` class is used. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Images in Bootstrap</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Image Example</h1>
    
</div>
</body>
</html>
```

The above code produces the following result, which shows an image with rounded corners.

Rounded Image Example



6.6.2 Circle Images

To reshape the image's structure as a circle, the *.rounded-circle* class is used. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Images in Bootstrap</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Image Example</h1>
    
</div>
</body>
</html>
```

The above code produces the following result, which shows an image in a circular form.

Circle Image Example



6.6.3 Thumbnail Image

To shape an image to a thumbnail, the *.img-thumbnail* is used. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Images in Bootstrap</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Image Example</h1>
    
</div>
</body>
</html>
```

The above code produces the following result, which shows an image in a thumbnail form.

Thumbnail Image Example

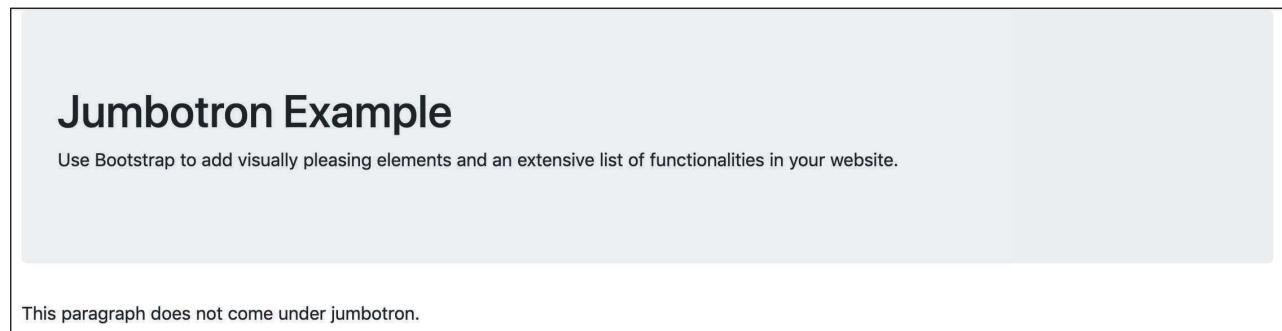


6.7 | Jumbotron

One of the most popular features of Bootstrap 4 is the “*jumbotron*”. It is basically a giant box with a gray background which is used to highlight importance of a piece of content. Jumbotron can accept other Bootstrap classes as well as majority of the HTML elements. To create a jumbotron, add a `<div>` element and place a `.jumbotron` class in it. Check the following example to generate a jumbotron:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Jumbotron</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <div class="jumbotron">
        <h1>Jumbotron Example</h1>
        <p>Use Bootstrap to add visually pleasing elements and an extensive list of functionalities in your website.</p>
    </div>
    <p>This paragraph does not come under jumbotron.</p>
</div>
</body>
</html>
```

The above code produces the following result, which shows the given text in a grey box.



To ensure that your jumbotron widens up to the complete width, use the `.jumbotron-fluid` class and add the container classes inside it. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Full-width Jumbotron</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="jumbotron jumbotron-fluid">
    <div class="container">
        <h1>Jumbotron Example</h1>
        <p>Use Bootstrap to add visually pleasing elements and an extensive list of functionalities in your website.</p>
    </div>
</div>
<div class="container">
    <h1>This heading does not come under jumbotron's content.</h1>
    <p>This paragraph does not come under jumbotron's content.</p>
</div>
</body>
</html>
```

The above code produces the following result, which shows the given text in a grey box which occupies the whole width of the viewport.

Jumbotron Fluid Example

Use Bootstrap to add visually pleasing elements and an extensive list of functionalities in your website.

This heading does not come under jumbotron's content.

This paragraph does not come under jumbotron's content.

QUICK CHALLENGE

Think of a real-life use case where you can use Jumbotron and write a code for it.

6.8 | Alerts



Bootstrap provides numerous types of alert boxes (or simply alerts) to display a special meaning. Alerts begin with the `.alert` class and are followed by contextual classes. Table 6.3 explains these contextual classes.

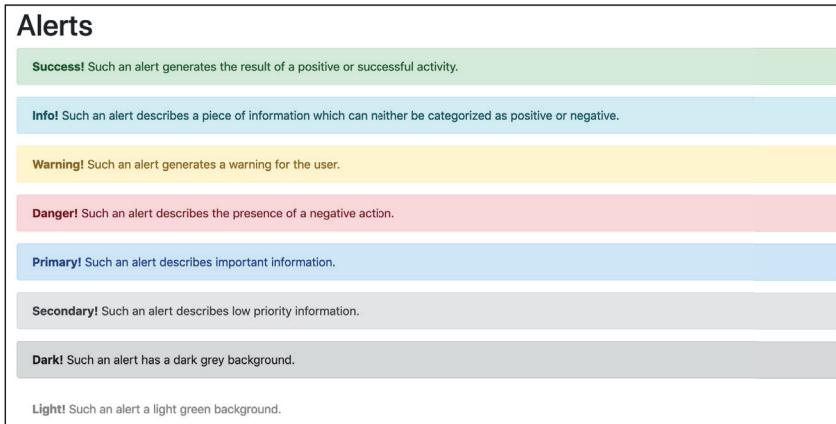
Table 6.3 Contextual classes

Code	Class
.alert-primary	It shows blue alert and specifies an important action.
.alert-secondary	It shows a grey alert and specifies a less important action.
.alert-success	It shows a green alert and specifies positive or successful action.
.alert-danger	It shows a red alert and specifies a potentially negative or dangerous action.
.alert-warning	It shows a yellow alert and specifies caution should be taken while dealing with this action.
.alert-info	It is a teal alert and it specifies a neutral action or informative change.
.alert-light	It is a light alert and it is shown by a light grey alert box.
.alert-dark	It is known as dark alert and it is represented by a dark grey alert box.

Consider the following example to check their use:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Bootstrap Alerts</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Alerts</h1>
    <div class="alert alert-success">
        <strong>Success!</strong> Such an alert generates the result of a positive or
        successful activity.
    </div>
    <div class="alert alert-info">
        <strong>Info!</strong> Such an alert describes a piece of information which can
        neither be categorized as positive or negative.
    </div>
    <div class="alert alert-warning">
        <strong>Warning!</strong> Such an alert generates a warning for the user.
    </div>
    <div class="alert alert-danger">
        <strong>Danger!</strong> Such an alert describes the presence of a negative
        action.
    </div>
    <div class="alert alert-primary">
        <strong>Primary!</strong> Such an alert describes important information.
    </div>
    <div class="alert alert-secondary">
        <strong>Secondary!</strong> Such an alert describes low priority information.
    </div>
    <div class="alert alert-dark">
        <strong>Dark!</strong> Such an alert has a dark grey background.
    </div>
    <div class="alert alert-light">
        <strong>Light!</strong> Such an alert has a light green background.
    </div>
</div>
</body>
</html>
```

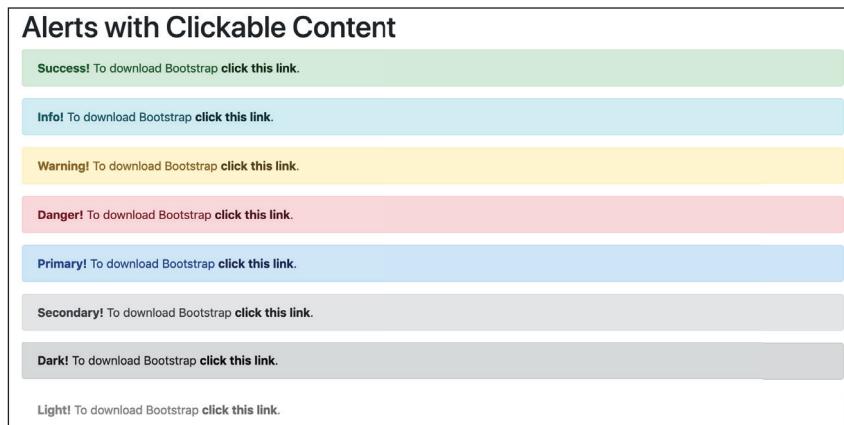
The above code produces the following result which shows various types of alert boxes.



It is also possible to add a link in alert boxes. This is done via the alert-link class. Consider the following example in which a link is added in each of the alert boxes.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Alert Links</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
    <h1>Alerts with Clickable Content</h1>
    <div class="alert alert-success">
        <strong>Success!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
    <div class="alert alert-info">
        <strong>Info!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
    <div class="alert alert-warning">
        <strong>Warning!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
    <div class="alert alert-danger">
        <strong>Danger!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
    <div class="alert alert-primary">
        <strong>Primary!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
    <div class="alert alert-secondary">
        <strong>Secondary!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
    <div class="alert alert-dark">
        <strong>Dark!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
    <div class="alert alert-light">
        <strong>Light!</strong> To download Bootstrap<a href="https://getbootstrap.com/" class="alert-link">click this link</a>.
    </div>
</div>
</body>
</html>
```

The above code produces the following result which shows various types of alert boxes with links inside.

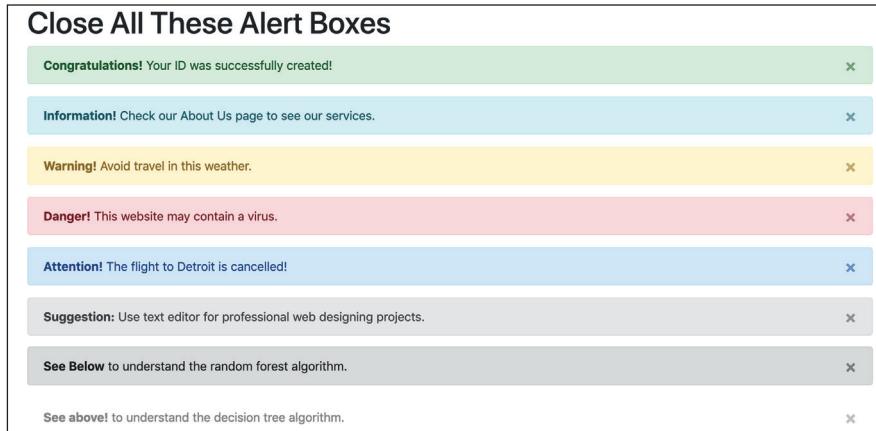


6.8.1 Closing Alerts

In order to display an alert which can be closed upon clicking, use the `.alert-dismissible` class in the container of the alert. Subsequently, place `class = "close"` and `data-dismiss = "alert"` on the content which you want to be disappear upon clicking. For example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Closing Alerts</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
    <h1>Close All These Alert Boxes</h1>
    <div class="alert alert-success alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Congratulations!</strong> Your ID was successfully created!
    </div>
    <div class="alert alert-info alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Information!</strong> Check our About Us page to see our services.
    </div>
    <div class="alert alert-warning alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Warning!</strong> Avoid travel in this weather.
    </div>
    <div class="alert alert-danger alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Danger!</strong> This website may contain a virus.
    </div>
    <div class="alert alert-primary alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Attention!</strong> The flight to Detroit is cancelled!
    </div>
    <div class="alert alert-secondary alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Suggestion:</strong> Use text editor for professional web designing projects.
    </div>
    <div class="alert alert-dark alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>See Below</strong> to understand the random forest algorithm.
    </div>
    <div class="alert alert-light alert-dismissible">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>See above!</strong> to understand the decision tree algorithm.
    </div>
</div></body>
</html>
```

The above code produces the following result, which shows all types of alerts with close buttons.

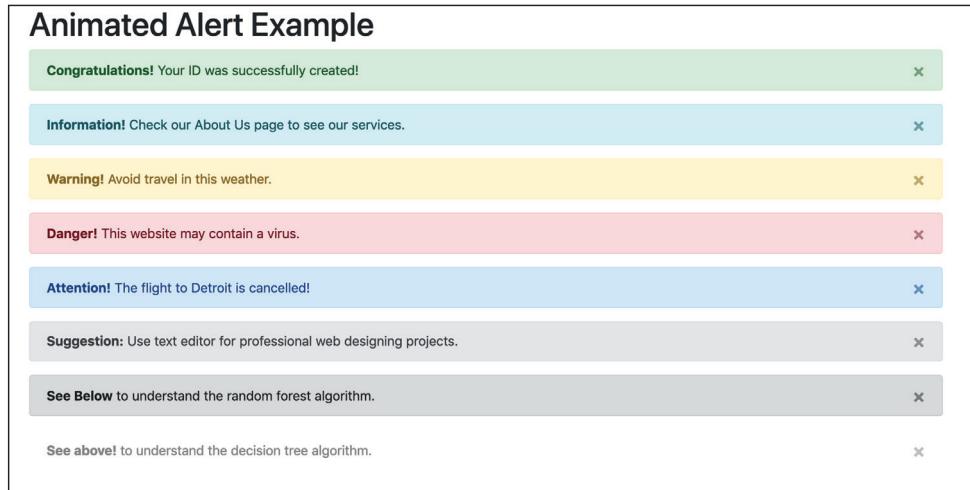


6.8.2 Animated Alerts

Bootstrap also allows us to add a little animation effect to the alert boxes. It provides two classes – *.fade* and *.show*. These will add fading effect when closing the pop-up. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Animated Alerts</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
    <h1>Animated Alert Examples</h1>
    <div class="alert alert-success alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Congratulations!</strong> Your ID was successfully created!
    </div>
    <div class="alert alert-info alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Information!</strong> Check our About Us page to see our services.
    </div>
    <div class="alert alert-warning alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Warning!</strong> Avoid travel in this weather.
    </div>
    <div class="alert alert-danger alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Danger!</strong> This website may contain a virus.
    </div>
    <div class="alert alert-primary alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Attention!</strong> The flight to Detroit is cancelled!
    </div>
    <div class="alert alert-secondary alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>Suggestion:</strong> Use text editor for professional web designing projects.
    </div>
    <div class="alert alert-dark alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>See Below</strong> to understand the random forest algorithm.
    </div>
    <div class="alert alert-light alert-dismissible fade show">
        <button type="button" class="close" data-dismiss="alert">&times;</button>
        <strong>See above!</strong> to understand the decision tree algorithm.
    </div>
</div>
</body>
</html>
```

The above program produces the following result. Upon clicking on the close button (x), the popup will disappear slowly.



How are these alerts better than plain JavaScript alert?

6.9 | Buttons

One of the most popular design elements of Bootstrap is its buttons; each of these buttons represents a certain type of action. In the following example we will explain how to create buttons:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Buttons</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Types of Button in Bootstrap</h1>
    <button type="button" class="btn">Simple Button</button>
    <button type="button" class="btn btn-success">Success Button</button>
    <button type="button" class="btn btn-warning">Warning Button</button>
    <button type="button" class="btn btn-danger">Danger Button</button>
    <button type="button" class="btn btn-primary">Primary </button>
    <button type="button" class="btn btn-secondary">Secondary Button</button>
    <button type="button" class="btn btn-link">Link Button</button>
    <button type="button" class="btn btn-dark">Dark Button</button>
    <button type="button" class="btn btn-info">Info Button</button>
    <button type="button" class="btn btn-light">Light Button</button>
</div>
</body>
</html>
```

The above code produces the following result, which shows all types of buttons that Bootstrap provides.



6.9.1 Outline and Size

Bootstrap 4 supports eight outline buttons and three basic sizes for buttons. Table 6.4 explains these eight outline buttons and three basis sizes.

Heading?	Heading?
.btn-outline-success	It is a grey bordered button and it specifies a positive or successful action.
.btn-outline-warning	It is an orange button and it represents a negative action or a warning.
.btn-outline-danger	It is a red bordered button and specifies a negative action or a danger.
.btn-outline-primary	It is a blue bordered button and specifies an important action.
.btn-outline-secondary	It is grey bordered button and specifies a less important action.
.btn-outline-dark	It is a dark grey bordered button
.btn-outline-info	It is a teal bordered button and it stands for a neutral action or informative change.
.btn-outline-light	It is a light grey bordered button.
.btn-lg	It is a large button. The "lg" stands for large.
.btn-md	It is a medium sized button. The "md" stands for medium.
.btn-sm	It is a small button. The "sm" stands for small.

Consider the following example to see all eight outline buttons and three basic sizes in action:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Buttons in Bootstrap</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Button Outline and Size</h2>
    <button type="button" class="btn btn-outline-success btn-lg">Success Button</button>
    <button type="button" class="btn btn-outline-warning btn-md">Warning Button</button>
    <button type="button" class="btn btn-outline-danger btn-sm">Danger Button</button>
    <button type="button" class="btn btn-outline-primary">Primary </button>
    <button type="button" class="btn btn-outline-secondary">Secondary Button</button>
    <button type="button" class="btn btn-outline-dark">Dark Button</button>
    <button type="button" class="btn btn-outline-info">Info Button</button>
    <button type="button" class="btn btn-outline-light">Light Button</button>
</div>
</body>
</html>
```

The above code produces the following result, which shows different button sizes with outline.



6.10 | Button Groups

Sometimes, there is a need for multiple buttons on a webpage. In such instances, it is unwise to create each button separately. Bootstrap 4 offers button groups through which multiple buttons can be created at once. This is done via the ".btn-group" in the <div> element. Consider the following example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Button Groups</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Grouped Buttons</h2>
    <p>By using the .btn-group, we can create display multiple buttons at once:</p>
    <div class="btn-group">
        <button type="button" class="btn btn-primary">Enter</button>
        <button type="button" class="btn btn-primary">Cancel</button>
        <button type="button" class="btn btn-primary">Reset</button>
    </div>
</div>
</body>
</html>
```

The above code produces the following result, which shows buttons in a group.



To adjust the size of buttons, you can use the following classes.

1. .btn-group-lg
2. .btn-group-md
3. .btn-group-sm

See the below example, which shows the group sizes in large, medium, and small.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Button Groups Sizes</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Grouped Buttons Sizes</h2>
    <h4>By using the .btn-group, we can create display multiple buttons at once:</h4><br/>
    <p>With btn-group-lg in large size:</p>
    <div class="btn-group btn-group-lg">
        <button type="button" class="btn btn-primary">Enter</button>
        <button type="button" class="btn btn-primary">Cancel</button>
        <button type="button" class="btn btn-primary">Reset</button>
    </div>
    <p>With btn-group-md in medium size:</p>
    <div class="btn-group btn-group-md">
        <button type="button" class="btn btn-primary">Enter</button>
        <button type="button" class="btn btn-primary">Cancel</button>
        <button type="button" class="btn btn-primary">Reset</button>
    </div>
    <p>With btn-group-md in small size:</p>
    <div class="btn-group btn-group-sm">
        <button type="button" class="btn btn-primary">Enter</button>
        <button type="button" class="btn btn-primary">Cancel</button>
        <button type="button" class="btn btn-primary">Reset</button>
    </div>
</div>
</body>
</html>
```

The above code produces the following result, which shows various groups with different sizes like large, medium, and small.

Grouped Buttons Sizes

By using the .btn-group, we can create display multiple buttons at once:

With btn-group-lg in large size:

Enter Cancel Reset

With btn-group-md in medium size:

Enter Cancel Reset

With btn-group-md in small size:

Enter Cancel Reset

6.10.1 Vertical

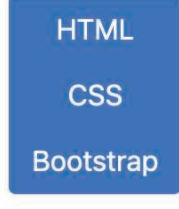
What if you want to display buttons in a vertical format? For that, you can use the `.btn-group vertical` class. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Bootstrap Button Classes</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
</script>
</head>
<body>
<div class="container">
<h2>Button Listed Vertically</h2>
<p>The .btn-group-vertical class is used to generate a vertical button group:</p>
<div class="btn-group-vertical">
<button type="button" class="btn btn-primary">HTML</button>
<button type="button" class="btn btn-primary">CSS</button>
<button type="button" class="btn btn-primary">Bootstrap</button>
</div>
</div>
</body>
</html>
```

The above code produces the following result, which shows buttons in vertical format.

Button Listed Vertically

The `.btn-group-vertical` class is used to generate a vertical button group:



HTML
CSS
Bootstrap

6.10.2 Dropdown Menus

You can nest button groups (add one group into another) to generate a dropdown menu. Button groups are inherently “inline”, which means that multiple button groups would appear side by side.

To click a button which opens the menu, go into its body and add the “`dropdown-toggle`”, which creates the dropdown on it and then add ‘`data-toggle="dropdown"`’ so it can open. For example:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Dropdown Menus</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Dropdown Menus</h2>
    <p>To create a dropdown menu add one button group inside another:</p>
    <div class="btn-group">
        <button type="button" class="btn btn-primary">Cybersecurity</button>
        <button type="button" class="btn btn-primary">Mobile App Development</button>
        <div class="btn-group">
            <button type="button" class="btn btn-primary dropdown-toggle" data-toggle="dropdown">
                Website Development
            </button>
            <div class="dropdown-menu">
                <a class="dropdown-item" href="#">HTML</a>
                <a class="dropdown-item" href="#">CSS</a>
                <a class="dropdown-item" href="#">Bootstrap</a>
            </div>
        </div>
    </div>
</div>
</body>
</html>

```

The above code produces the following result, which shows a dropdown menu.

Dropdown Menus

To create a dropdown menu add one button group inside another:

Cybersecurity
Mobile App Development
Website Development ▾

HTML
CSS
Bootstrap

6.11 | Progress Bars



Progress bars are used to display the remaining portion of an online activity. They can be created by using the `.progress` class in the container after which the `".progress-bar"` class is placed in the element of the container. In order to define the “*progress*” of the bar, apply a percentage through the CSS width property. For a simple progress bar use the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Progress Bar</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
<h2>A Simple Progress Bar</h2>
<div class="progress">
    <div class="progress-bar" style="width:50%"></div>
</div>
</div>
</body></html>
```

The above code produces the following result, which shows a progress bar.

A Simple Progress Bar



6.11.1 Height and Label

It is possible to resize the height of a progress bar. To do this, use the “*height*” property. Note that this height must be set same for two elements: container and the progress bar. To show the percentage of the bar, simply add the value of the width in its *<div>* tag. Check the following example:

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Progress Bar</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"></script>
</head>
<body>
<div class="container">
<h2>Adjusting the Height of the Progress Bar!</h2>
<div class="progress" style="height:50px">
    <div class="progress-bar" style="width:80%;height:50px">80%</div>
</div>
<br>
<div class="progress" style="height:25px">
    <div class="progress-bar" style="width:60%;height:25px">60%</div>
</div>
<br>
<div class="progress" style="height:10px">
    <div class="progress-bar" style="width:40%;height:10px">40%</div>
</div>
</div>
</body></html>
```

The above code produces the following result, which shows a progress bar with height and label.

Adjusting the Height of the Progress Bar!



6.11.2 Adding Colors to Progress Bar

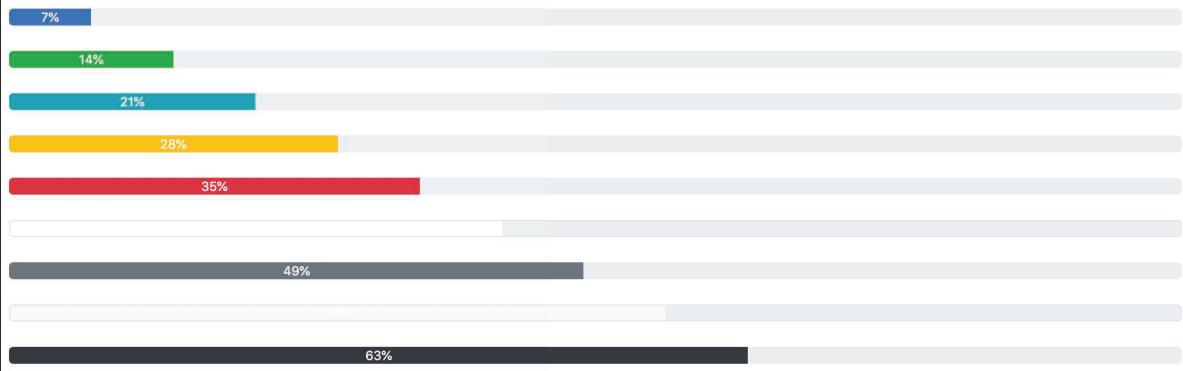
By default, Bootstrap's progress bar is primary. However, it can be modified to accept any of the color contextual classes. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Progress Bars</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Adding Colors in the Progress Bars</h2>
    <p>Contextual color classes can be used with the progress bars:</p>
    <div class="progress">
        <div class="progress-bar" style="width:7%">7%</div>
    </div><br>
    <div class="progress">
        <div class="progress-bar bg-success" style="width:14%">14%</div>
    </div><br>
    <div class="progress">
        <div class="progress-bar bg-info" style="width:21%">21%</div>
    </div><br>
    <div class="progress">
        <div class="progress-bar bg-warning" style="width:28%">28%</div>
    </div><br>
    <div class="progress">
        <div class="progress-bar bg-danger" style="width:35%">35%</div>
    </div><br>
    <div class="progress border">
        <div class="progress-bar bg-white" style="width:42%">42%</div>
    </div><br>
    <div class="progress">
        <div class="progress-bar bg-secondary" style="width:49%">49%</div>
    </div><br>
    <div class="progress border">
        <div class="progress-bar bg-light" style="width:56%">56%</div>
    </div><br>
    <div class="progress">
        <div class="progress-bar bg-dark" style="width:63%">63%</div>
    </div>
</div>
</body>
</html>
```

The above code produces the following result, which shows a progress bar with colors.

Adding Colors in the Progress Bars

Contextual color classes can be used with the progress bars:



6.11.3 Adding Stripes to Progress Bars

By utilizing the “`.progress-bar-striped`” class, it is possible to incorporate stripes in your progress bars. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Striped Progress Bars</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Progress Bars with Stripes</h1>
    <div class="progress">
        <div class="progress-bar progress-bar-striped" style="width:10%">10%</div>
    </div>
    <br>
    <div class="progress">
        <div class="progress-bar bg-success progress-bar-striped" style="width:20%">20%</div>
    </div>
    <br>
    <div class="progress">
        <div class="progress-bar bg-info progress-bar-striped" style="width:30%">30%</div>
    </div>
    <br>
    <div class="progress">
        <div class="progress-bar bg-warning progress-bar-striped" style="width:40%">40%</div>
    </div>
    <br>
    <div class="progress">
        <div class="progress-bar bg-danger progress-bar-striped" style="width:40%">40%</div>
    </div>
</div>
</body></html>
```

The above code produces the following result, which shows a progress bar with stripes.

Progress Bars with Stripes



6.11.4 Adding Animations to the Progress Bar

You must have seen that “lively” animated progress in many websites. To replicate it use the “*.progress-bar-animated*” class. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Animated Progress Bar</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h2>Lively Progress Bar</h2>
    <div class="progress">
        <div class="progress-bar progress-bar-striped progress-bar-animated" style="width:90%">Loading</div>
    </div>
</div>
</body>
</html>
```

The above code produces the following result, which shows a progress with animation.

Lively Progress Bar

A screenshot showing a single horizontal progress bar with the word "Loading" in the center. The bar has a blue striped pattern.



6.12 | Pagination

When the number of webpages in a website increases, pagination can be implemented using Bootstrap. To apply pagination, the `.pagination` class is used in an unordered list while the `.page-item` and `.page-link` classes are used with the list items. For example,

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Pagination</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Pagination Example</h1>
    <ul class="pagination">
        <li class="page-item" ><a class="page-link" href="#">Last</a></li>
        <li class="page-item" ><a class="page-link" href="#">1</a></li>
        <li class="page-item" ><a class="page-link" href="#">2</a></li>
        <li class="page-item" ><a class="page-link" href="#">3</a></li>
        <li class="page-item" ><a class="page-link" href="#">4</a></li>
        <li class="page-item" ><a class="page-link" href="#">5</a></li>
        <li class="page-item" ><a class="page-link" href="#">Next</a></li>
    </ul>
</div>
</body>
</html>
```

The above code produces the following result, which shows pagination.

Pagination Example

Last	1	2	3	4	5	Next
------	---	---	---	---	---	------

To show the current page, an active class can be used. To modify the size, simply add “*pagination-lg*”, “*pagination-md*” or “*pagination-sm*”. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Pagination Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Active State and Size in Pagination</h1>
    <h3>Pagination in Large Size:</h3>
    <ul class="pagination pagination-lg">
        <li class="page-item"><a class="page-link" href="#">Last</a></li>
        <li class="page-item"><a class="page-link" href="#">1</a></li>
        <li class="page-item active"><a class="page-link" href="#">2</a></li>
        <li class="page-item"><a class="page-link" href="#">3</a></li>
        <li class="page-item"><a class="page-link" href="#">4</a></li>
        <li class="page-item"><a class="page-link" href="#">5</a></li>
        <li class="page-item"><a class="page-link" href="#">Next</a></li>
    </ul>
    <br />
    <h3>Pagination in Medium Size:</h3>
    <ul class="pagination pagination-md">
        <li class="page-item"><a class="page-link" href="#">Last</a></li>
        <li class="page-item"><a class="page-link" href="#">1</a></li>
        <li class="page-item active"><a class="page-link" href="#">2</a></li>
        <li class="page-item"><a class="page-link" href="#">3</a></li>
        <li class="page-item"><a class="page-link" href="#">4</a></li>
        <li class="page-item"><a class="page-link" href="#">5</a></li>
        <li class="page-item"><a class="page-link" href="#">Next</a></li>
    </ul>
    <br />
    <h3>Pagination in Small Size:</h3>
    <ul class="pagination pagination-sm">
        <li class="page-item"><a class="page-link" href="#">Last</a></li>
        <li class="page-item"><a class="page-link" href="#">1</a></li>
        <li class="page-item active"><a class="page-link" href="#">2</a></li>
        <li class="page-item"><a class="page-link" href="#">3</a></li>
        <li class="page-item"><a class="page-link" href="#">4</a></li>
        <li class="page-item"><a class="page-link" href="#">5</a></li>
        <li class="page-item"><a class="page-link" href="#">Next</a></li>
    </ul>
</div>
</body>
</html>
```

The above code produces the following result, which shows pagination with state like current page and sizes.

Active State and Size in Pagination

Pagination in Large Size:

Last	1	2	3	4	5	Next
------	---	---	---	---	---	------

Pagination in Medium Size:

Last	1	2	3	4	5	Next
------	---	---	---	---	---	------

Pagination in Small Size:

Last	1	2	3	4	5	Next
------	---	---	---	---	---	------

6.13 | Cards



Bootstrap's card is a box which is covered by some padding and borders. It can contain content, colors, headers, and footers. It is generated by using the ".class" class while its contents are specified by using the ".card-body" class. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Cards</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
<h1>A Simple Example</h1>
<div class="card">
    <div class="card-body">First Card</div>
</div>
<div class="card">
    <div class="card-body">Second Card</div>
</div>
</div>
</body>
</html>
```

The above code produces the following result, which shows cards.

A Simple Card Example

First Card

Second Card

Headers and footers can be placed in the card through the use of “*.card-header*” and “*.card-footer*” classes, respectively.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Cards</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Card with Header and Footer Example</h1>
    <div class="card">
        <div class="card-header">Header</div>
        <div class="card-body">Body</div>
        <div class="card-footer">Footer</div>
    </div>
</div>
</body>
</html>
```

The above code produces the following result, which shows cards with header and footer.

Card with Header and Footer Example

Header

Body

Footer

6.14 | Navigation Menus

To generate a standard navigation menu which is horizontal in nature, use the “*.nav*” class in an unordered list along with “*.nav-item*” and “*.nav-link*” classes in its contents (list items). For example:



```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Nav Menus</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Nav Menu Example</h1>
    <ul class="nav">
        <li class="nav-item">
            <a class="nav-link" href="#">First Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Second Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Third Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Fourth Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Fifth Item</a>
        </li>
    </ul>
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with navigation menu.

Nav Menu Example

First Item	Second Item	Third Item	Fourth Item	Fifth Item
----------------------------	-----------------------------	----------------------------	-----------------------------	----------------------------

6.14.1 Vertical Nav Menus

To convert the above-mentioned horizontal menu in a vertical navigation menu, simply add “*flex column*” ahead of the “*nav*” class. For example:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Nav Menus</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Nav Menu Example</h1>
    <ul class="nav flex-column">
        <li class="nav-item">
            <a class="nav-link" href="#">First Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Second Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Third Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Fourth Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Fifth Item</a>
        </li>
    </ul>
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with vertical navigation menu.

Vertical Nav Menu Example

- [First Item](#)
- [Second Item](#)
- [Third Item](#)
- [Fourth Item](#)
- [Fifth Item](#)

Navigation menu can be used to create a main menu or submenus for easy navigation.

6.14.2 Tabs

Now in order to change the navigation menu into one which has tabs, you can use the “`.nav-tabs`” class. For instance

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Nav Menus</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Nav Menu Example</h1>
    <ul class="nav nav-tabs">
        <li class="nav-item">
            <a class="nav-link active" href="#">First Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Second Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Third Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Fourth Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Fifth Item</a>
        </li>
    </ul>
</div>
</body>
</html>
```

The above code produces the following result, which shows a page with tabs menu.

Nav Tabs Example

First Item	Second Item	Third Item	Fourth Item	Fifth Item
------------	-------------	------------	-------------	------------

6.14.3 Pills

Similarly, you can use the “`.nav-pills`” class to add pills in the navigation menu. Pills are simple navigation components which offers various different layouts to enable faster navigation between options.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Nav Menus</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Nav Menu Example</h1>
    <ul class="nav nav-pills">
        <li class="nav-item">
            <a class="nav-link active" href="#">First Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Second Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Third Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Fourth Item</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Fifth Item</a>
        </li>
    </ul>
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with navigation pills menu.

Navigation Pills Example

[First Item](#) [Second Item](#) [Third Item](#) [Fourth Item](#) [Fifth Item](#)

6.15 | Navigation Bar

A navigation bar consists of a header which is positioned on the top of a webpage. In Bootstrap, the functionality of such navigation bars relies on the size of user screens – it can extend or collapse accordingly.

To create a navigation bar, use the `.navbar` class while the `responsiveness` can be configured through the `".navbar-expand-xl|lg|md|sm"` class. For example:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Navigation Bars</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<nav class="navbar navbar-expand-sm bg-secondary">
    <ul class="navbar-nav">
        <li class="nav-item">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">About</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Careers</a>
        </li>
    </ul>
</nav>
<br>
<div class="container-fluid">
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with navigation bar menu.

Navigation Bar Example

[Home](#) [About](#) [Services](#) [Careers](#)

To generate a vertical navigation bar, simply eliminate the “`.navbar-expand-xl|lg|md|sm`” class. For getting your navigation bar in the middle of the page, use the “`justify-content-center`” class.

6.15.1 Colorful Navigation Bars

All the background color classes can be used to modify the navigation bar’s background colors. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Colorful Navigation Bars</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Navigation Bars</h1>
</div>
<nav class="navbar navbar-expand-sm bg-light navbar-light justify-content-center">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Vision</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Careers</a>
        </li>
    </ul>
</nav>
<nav class="navbar navbar-expand-sm bg-secondary navbar-dark">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Vision</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Careers</a>
        </li>
    </ul>
</nav>
<nav class="navbar navbar-expand-sm bg-primary navbar-dark">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Vision</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
    </ul>
</nav>
```

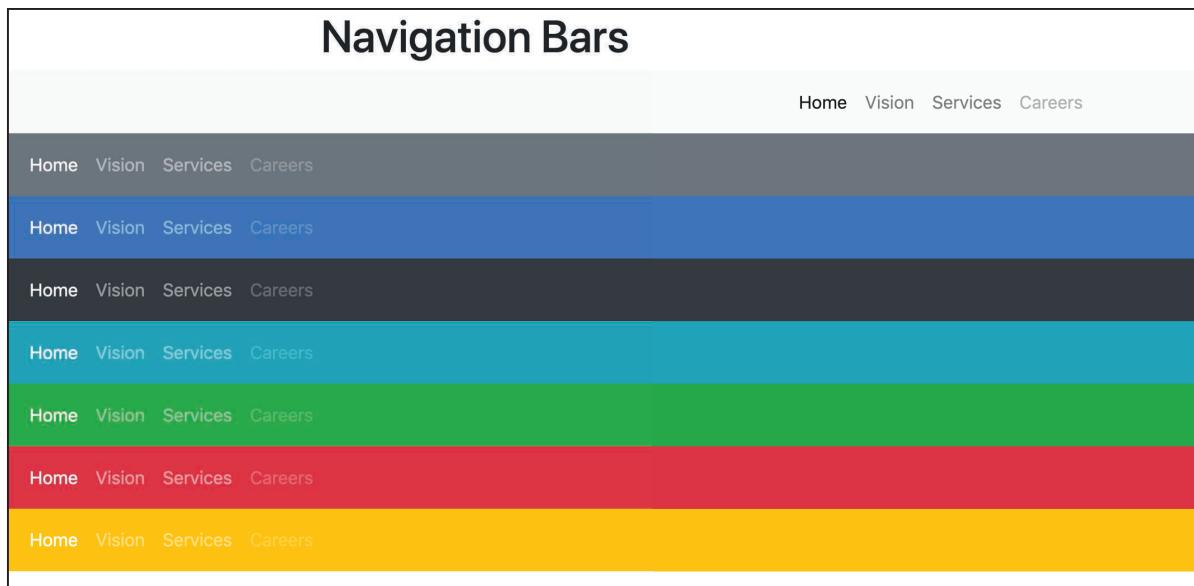
```
<li class="nav-item">
    <a class="nav-link disabled" href="#">Careers</a>
</li>
</ul>
</nav>
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Vision</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Careers</a>
        </li>
    </ul>
</nav>
<nav class="navbar navbar-expand-sm bg-info navbar-dark">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Vision</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Careers</a>
        </li>
    </ul>
</nav>
<nav class="navbar navbar-expand-sm bg-success navbar-dark">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Vision</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Careers</a>
        </li>
    </ul>
</nav>
<nav class="navbar navbar-expand-sm bg-danger navbar-dark">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
```

```

<li class="nav-item">
    <a class="nav-link" href="#">Vision</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="#">Services</a>
</li>
<li class="nav-item">
    <a class="nav-link disabled" href="#">Careers</a>
</li>
</ul>
</nav>
<nav class="navbar navbar-expand-sm bg-warning navbar-dark">
    <ul class="navbar-nav">
        <li class="nav-item active">
            <a class="nav-link" href="#">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Vision</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="#">Services</a>
        </li>
        <li class="nav-item">
            <a class="nav-link disabled" href="#">Careers</a>
        </li>
    </ul>
</nav>
</body>
</html>

```

The above code produces the following result, which shows a page with colorful navigation bar.



6.16 | Forms

Bootstrap offers two types of layouts for forms. First, we have a form with full width which is referred to as the stacked form. Second is an inline form.



6.16.1 Stacked Form

To generate a stacked form which contains two fields for input and a checkbox and submit buttons each, we can make use of the “*form-group*” class. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Stacked Forms</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>How to Create Stacked Form</h1>
    <form action="/action_page.php">
        <div class="form-group">
            <label for="email">Email Address</label>
            <input type="email" class="form-control" id="email" placeholder="Type Your Email Address" name="email">
        </div>
        <div class="form-group">
            <label for="pwd">Password</label>
            <input type="password" class="form-control" id="pwd" placeholder="Type Your Password" name="password">
        </div>
        <div class="form-group form-check">
            <label class="form-check-label">
                <input class="form-check-input" type="checkbox" name="Logged in"> Logged in
            </label>
        </div>
        <button type="submit" class="btn btn-primary">Proceed</button>
    </form>
</div>
</body>
</html>
```

The above code produces the following result, which shows a page with stacked form.

6.16.2 Bootstrap Inline Form

An inline form is one in which all the form elements are aligned on the left side along with being “*inline*” by default. In the following example, we generate an inline form:

```

<!DOCTYPE html>
<html lang="en">
<head>
<title>Inline Form</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js">
</script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
</script>
</head>
<body>
<div class="container">
    <h1>How to Create an Inline Form in Bootstrap</h1>
    <form class="form-inline" action="/action_page.php">
        <label for="email">Email Address</label>
        <input type="email" class="form-control" id="email" placeholder="Type Your Email Address" name="email address">
        <label for="pwd">Password</label>
        <input type="password" class="form-control" id="pw" placeholder="Type Your Password" name="password">
        <div class="form-check">
            <label class="form-check-label">
                <input class="form-check-input" type="checkbox" name="logged in" checked="checked" /> Logged In
            </label>
        </div>
        <button type="submit" class="btn btn-primary">Proceed</button>
    </form>
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with inline form.

How to Create an Inline Form in Bootstrap

Email Address Password Logged In

6.17 | Carousel



Have you ever seen those images on websites which rotate one-by-one as a slideshow? If you are aiming to add such functionality in your website, then Bootstrap's highly powerful carousel class is the way to go.

Bootstrap's carousel is often used in website as a slideshow for checking multiple elements. While creating the carousel, you have to first set the number of “*indicators*” which would require going through the images in the slideshow. For example, in the example below we used 4. This is done by using the “*carousel-indicators*” class which is placed in an unordered list. For the list items, the “*data-target*” class is used.

Afterward, the images in the slideshow have to be specified by using the “*carousel-inner*” class in a *div* tag. Images are set through the “*carousel-item*” class.

In the end, controls are added through “*carousel-control-prev*” and “*carousel-control-next*” class. As their name suggests, they are used to go back and forth in the slideshow. Note that by default the slideshow would run itself. To understand carousel, consider the following example:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Carousel Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
    <style>
        .carousel-inner img {
            width: 100%;
            height: 100%;
        }
    </style>
</head>
<body>
<div id="demo" class="carousel slide" data-ride="carousel">
    <ul class="carousel-indicators">
        <li data-target="#demo" data-slide-to="0" class="active"></li>
        <li data-target="#demo" data-slide-to="1"></li>
        <li data-target="#demo" data-slide-to="2"></li>
        <li data-target="#demo" data-slide-to="3"></li>
    </ul>
    <div class="carousel-inner">
        <div class="carousel-item active">
            
        </div>
        <div class="carousel-item">
            
        </div>
        <div class="carousel-item">
            
        </div>
        <div class="carousel-item">
            
        </div>
        <div class="carousel-item">
        </div>
        <a class="carousel-control-prev" href="#demo" data-slide="prev">
            <span class="carousel-control-prev-icon"></span>
        </a>
        <a class="carousel-control-next" href="#demo" data-slide="next">
            <span class="carousel-control-next-icon"></span>
        </a>
    </div>
</body>
</html>

```

The above code produces the following result, which shows a page with carousel like sliding images. It also has two buttons to go to previous or next image.

Carousel Example



It is also possible to add captions inside the slideshow by using the “`<div class = "carousel-caption">`” class. For example:

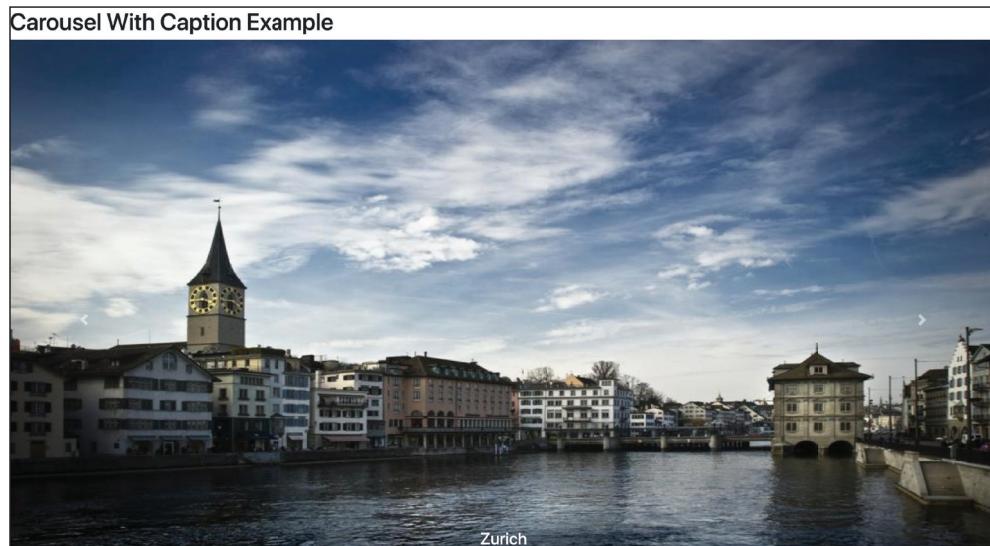
```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Carousel With Caption Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
    <style>
        .carousel-inner img {
            width: 100%;
            height: 100%;
        }
    </style>
</head>
<body>
<div id="demo" class="carousel slide" data-ride="carousel">
    <h1>Carousel With Caption Example</h1>
    <ul class="carousel-indicators">
        <li data-target="#demo" data-slide-to="0" class="active"></li>
        <li data-target="#demo" data-slide-to="1"></li>
        <li data-target="#demo" data-slide-to="2"></li>
        <li data-target="#demo" data-slide-to="3"></li>
    </ul>
    <div class="carousel-inner">
        <div class="carousel-item active">
            
            <div class="carousel-caption">
                <h3>London</h3>
                <p>London is quite expensive but it is worth it!</p>
            </div>
        </div>
        <div class="carousel-item">
            
        </div>
    </div>
</div>
```

```

<div class="carousel-caption">
    <h3>Berlin</h3>
    <p>Berlin is one of the biggest tech hubs in the world.</p>
</div>
</div>
<div class="carousel-item">
    
    <div class="carousel-caption">
        <h3>Zurich</h3>
        <p>Zurich is the center of banking and finance.</p>
    </div>
</div>
<div class="carousel-item">
    
    <div class="carousel-caption">
        <h3>Venice</h3>
        <p>Few cities can claim to match Venice in beauty.</p>
    </div>
</div>
<div class="carousel-item">
    <a class="carousel-control-prev" href="#demo" data-slide="prev">
        <span class="carousel-control-prev-icon"></span>
    </a>
    <a class="carousel-control-next" href="#demo" data-slide="next">
        <span class="carousel-control-next-icon"></span>
    </a>
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with carousel like sliding images with caption at the bottom. It also has two buttons to go to previous or next image.





6.18 | Media Objects

Bootstrap is known for providing a simple method to align media objects such as images or videos along with content. These objects could be used for various types of components like blog comments, tweets, etc. They can feature a left- or right-aligned image along with the textual content. The media makes it simpler to construct complex and repetitive components where some type of media is placed along with content that does not wrap around the mentioned media. The main aim of the object is to keep the code for building these blocks of information very short.

.media: This class makes it possible to float a media object (e.g., video, image, or audio) to the left or right side of a content block.

See the following simple media object example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Media Object Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Media Object Example</h1>
    <div class="media">
        
        <div class="media-body">
            <h5 class="mt-0">Mayur Ramgir</h5>
            Every "Failure" brings new "Opportunity"
            Every "Darkness" brings new "Light"
            Every "Blindness" brings new "Vision"
            Every "No" brings new "Yes"
            Every "Closed Door" opens up a new door
            There is no such thing called "End of the World". Keep fighting the good fight my friend. One day would be yours, if not today, it will be tomorrow.
        </div>
    </div>
</div>
</body>
</html>
```

The above code produces the following result, which shows a page with media object.

Media Object Example



Mayur Ramgir

Every "Failure" brings new "Opportunity" Every "Darkness" brings new "Light" Every "Blindness" brings new "Vision" Every "No" brings new "Yes" Every "Closed Door" opens up a new door There is no such thing called "End of the World". Keep fighting the good fight my friend. One day would be yours, if not today, it will be tomorrow.

Building nested media objects: you can also nest media objects inside other media objects. It can be used for creating comment threads in a blog post. See the following example:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Media Object Nested Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Media Object Nested Example</h1>
    <ul class = "media-list">
        <li class = "media">
            <a class = "pull-left" href = "#">
                
            </a>
            <div class = "media-body">
                <h4 class = "media-heading">Mayur Ramgir</h4>
                <p>
                    Trust is the most important element. By con you can win once and lose forever but with honesty you can win over and over again.
                </p>
                <div class = "media">
                    <a class = "pull-left" href = "#">
                        
                    </a>
                    <div class = "media-body">
                        <h4 class = "media-heading">Mayur Ramgir</h4>
                        World is so beautiful when you look beyond your self-interest.
                        Success and self do not work together. Rise up the ordinary.
                    <div class = "media">
                        <a class = "pull-left" href = "#">
                            
                        </a>
                        <div class = "media-body">
                            <h4 class = "media-heading">Mayur Ramgir</h4>
                            The greatest thing in the world is not in dying rich but leaving the richness behind for the world to treasure.
                        </div>
                    </div>
                </div>
            </div>
        <div class = "media">
            <a class = "pull-left" href = "#">
                
            </a>
            <div class = "media-body">
                <h4 class = "media-heading">Mayur Ramgir</h4>
                Dreams are dreams until you make them a reality. And that won't happen until you give up your sleep, Wake Up.
            </div>
        </div>
    </li>
</ul>
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with nested media objects.

Media Object Nested Example

Mayur Ramgir
Trust is the most important element. By con you can win once and lose forever but with honesty you can win over and over again.

Mayur Ramgir
World is so beautiful when you look beyond your self-interest. Success and self do not work together. Rise up the ordinary.

Mayur Ramgir
The greatest thing in the world is not in dying rich but leaving the richness behind for the world to treasure.

Mayur Ramgir
Dreams are dreams until you make them a reality. And that won't happen until you give up your sleep, Wake Up.

The alignment of media objects: By simply tweaking the HTML code itself, we can easily change the horizontal alignment media and content. Apart from this, you can also align images or other media files at the centre or bottom of the content block by utilizing the flexbox utility classes.

See the following example which shows alignment of media objects:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Media Object Alignment Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Media Object Alignment Example</h1>
    <div class="media">
        
        <div class="media-body">
            <h5 class="mt-0">Mayur Ramgir</h5>
            Don't waste your time competing with someone. Instead focus on competing with yourself. Leaders don't compete, followers do.
        </div>
    </div>
    <br /><br />
    <div class="media">
        
        <div class="media-body">
            <h5 class="mt-0">Mayur Ramgir</h5>
            Spread your wings, become a role model for others to follow so that they can match you step for step in their own quest to achieve success.
        </div>
    </div>
    <br /><br />
    <div class="media">
        
        <div class="media-body">
            <h5 class="mt-0">Mayur Ramgir</h5>
            Opportunity to help others is the opportunity to help yourself in disguise.
        </div>
    </div>
</div>
</body>
</html>
```

The above code produces the following result, which shows various alignment of media objects.

Media Object Alignment Example

 Mayur Ramgir

Don't waste your time competing with someone. Instead focus on competing with yourself. Leaders don't compete, followers do.

 Mayur Ramgir

Spread your wings, become a role model for others to follow so that they can match you step by step in their own quest to achieve success.

 Mayur Ramgir

Opportunity to help others is the opportunity to help yourself in disguise.

Creating a media list: We can create a media list by using the media object classes and HTML list elements. This can also be done by placing media objects inside a list. These lists are very resourceful in the representation of user comment threads, lists, articles, etc.

See the following a simple media object list example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Media Object List Example</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.6/dist/umd/popper.min.js">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js">
    </script>
</head>
<body>
<div class="container">
    <h1>Media Object List Example</h1>
    <ul class = "media-list">
        <li class = "media">
            <a class = "pull-left" href = "#">
                
            </a>
            <div class = "media-body">
                <h4 class = "media-heading">Mayur Ramgir</h4>
                <p>
                    Fools are those who do not understand, every action has an equal and opposite reaction. Take your actions carefully to avoid unwanted results.
                </p>
            </div>
        </li>
        <li class = "media">
            <a class = "pull-left" href = "#">
                
            </a>
            <div class = "media-body">
                <h4 class = "media-heading">Mayur Ramgir</h4>
```

```

<p>
    Experience every failure closely because each failure is taking you
    closer to your goal and preparing you to be the unbeatable.
</p>
</div>
</li>
<li class = "media">
    <a class = "pull-left" href = "#">
        
    </a>
    <div class = "media-body">
        <h4 class = "media-heading">Mayur Ramgir</h4>
        <p>
            Explore new avenues and look for ways to have a greater, much more
            influential impact on the world around you.
        </p>
    </div>
</li>
</ul>
</div>
</body>
</html>

```

The above code produces the following result, which shows a page with media object list.

Media Object List Example



Mayur Ramgir

Fools are those who do not understand, every action has an equal and opposite reaction. Take your actions carefully to avoid unwanted results.



Mayur Ramgir

Experience every failure closely because each failure is taking you closer to your goal and preparing you to be the unbeatable.



Mayur Ramgir

Explore new avenues and look for ways to have a greater, much more influential impact on the world around you.



What is media object and what is its use?

Summary

In a short period of time, Bootstrap has become one of the leading front-end web technologies out there. In this chapter, we presented and practiced the most common and useful features of Bootstrap. This chapter would not only facilitate you in creating more elegant and sleek websites, but it can also serve as a basis for you to learn more complex front-end technologies.

In this chapter, we have learned the following concepts:

1. Bootstrap and its use to render content properly on all the screens.
2. Container and types of containers.
3. Grid and its use.
4. Various elements such as tables, forms, etc. and how to make sure they render properly.
5. Visually appealing features such as progress bar, navigation bar, etc.

In the Chapter 7, we will learn how to build HTML pages for the project idea we have defined in Chapter 2. We will also discuss how to create pages based on the designs and flow chart we have created.

Multiple-Choice Questions

1. What is the global default line-height in Bootstrap?
 - (a) 1.8
 - (b) 1.428
 - (c) 1.228
 - (d) 1.128
2. The main use of Glyphicons is _____.
 - (a) Graphic images
 - (b) Providing different icons
 - (c) Animation
 - (d) Slideshow
3. What kind of layout in Bootstrap is used for providing 100% width?
 - (a) Fixed layout
 - (b) Fluid layout
4. Which one of the given classes can make a thumbnail image?
 - (a) .img-thumbnail-image
 - (b) .img-thumb
 - (c) .img-thumbnail
 - (d) .img-tmbnail
5. Which one of the following plugins is utilized to create a modal window?
 - (a) window
 - (b) popup
 - (c) dialogbox
 - (d) modal

Review Questions

1. What is the use of Bootstrap?
2. What are the prerequisites for using Bootstrap?
3. How can you use the Bootstrap navbar?
4. What are Bootstrap Cards?
5. How can you use Form?
6. How can you use progress bar and set colors?
7. What is Jumbotron and how can one use it?
8. How can you use alerts?
9. What is pagination and how can you use it?
10. What are the key components of Bootstrap?

Exercises

1. Create a page and add table. Make sure the table is responsive and renders nicely on all types of screen sizes.
2. Create a registration page and add fields like name, address, etc. to capture data. Also add alerts if user is going to close the window.
3. Create a page and add file upload functionality. Then add a progress bar so that you can show the progress of file upload.
4. Create a page to show data in a graphical form. Use bar and pie charts to show the last 7 days weather data.
5. Create a page and add Grid to show stock prices data of Apple.

Project Ideas

Project 1: Take the dictionary project idea from Chapter 5 and make sure all the pages that we have created so far are rendering properly on all types of screen sizes. Add Bootstrap to this code to make it responsive. Also add additional features like progress bar, alerts, etc. to make sure the elements look good.

Add a new page where users can search for a particular word. Make sure we have nice effects and animation for this search feature.

Project 2: Create your own resume page in HTML, CSS, and Bootstrap. Add your photo and all the courses you have taken so far in schools/colleges/universities. Use different colors, different font styles, and most importantly make sure the page looks good on various different devices.

Recommended Readings

1. Benjamin Jakobus. 2018. *Mastering Bootstrap 4: Master the latest version of Bootstrap 4 to build highly customized responsive web apps, Second Edition*. Packt Publishing Ltd: Birmingham
2. Eduonix Learning Solutions. 2018. *Learning Bootstrap 4 by Building Projects: Develop 5 real-world Bootstrap 4.x projects from scratch*. Packt Publishing Ltd: Birmingham
3. Jacob D Lett. 2018. *Bootstrap 4 Quick Start: A Beginner's Guide to Building Responsive Layouts with Bootstrap 4*. Bootstrap Creative: Michigan
4. Bootstrap – <https://getbootstrap.com/>
5. W3School Bootstrap 3 – <https://www.w3schools.com/bootstrap/>
6. W3School Bootstrap 4 – <https://www.w3schools.com/bootstrap4/>

Build Pages for MyEShop with HTML and CSS

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Setup and integration of elements to create HTML pages.
- Entity relationship diagram.
- Development of web application using HTML and CSS.
- Use of any text editor to create HTML pages.
- How to add CSS on HTML elements.

7.1 | Setting up Environment

For HTML and CSS you do not really need to setup any environment. You can use any text editor you like. You can use the same Eclipse editor you have used in Chapter 3.

7.2 | Identify the Pages

Let us explore our entity relationship diagram again. See Figure 7.1 which we have presented in Chapter 2 (Figure 2.1) when we worked on architectural designs.

As per the entity relationship diagram (ERD), we have identified at least the following main entities:

Customer	Payment	Inventory
Address	Product	Catalogue
Shipping	Currency	Warehouse
Invoice	Shopping Cart	Vendor
Offer		

We have used these entities to create a page flow diagram in Chapter 2 (Figure 2.8); it is reproduced in Figure 7.2 which shows order flow for ease of understanding.

Figure 7.2 gives us an idea about which pages we should create for one particular use case. There are multiple use cases you can think of in the system. According to this use case, the flow starts from a main webpage, that is, the home page of *MyEShop* site. This page has astore-like look and feel. It also allows users to go to the login page or register page in order to access their personal dashboard. Now let us first start building the home page.

7.3 | Getting Started with HTML Pages

In this section, we will create a page which the user will visit as soon as he/she hits the website URL in a browser window. For this we will use a text editor and save this file with the name “*home.html*”.



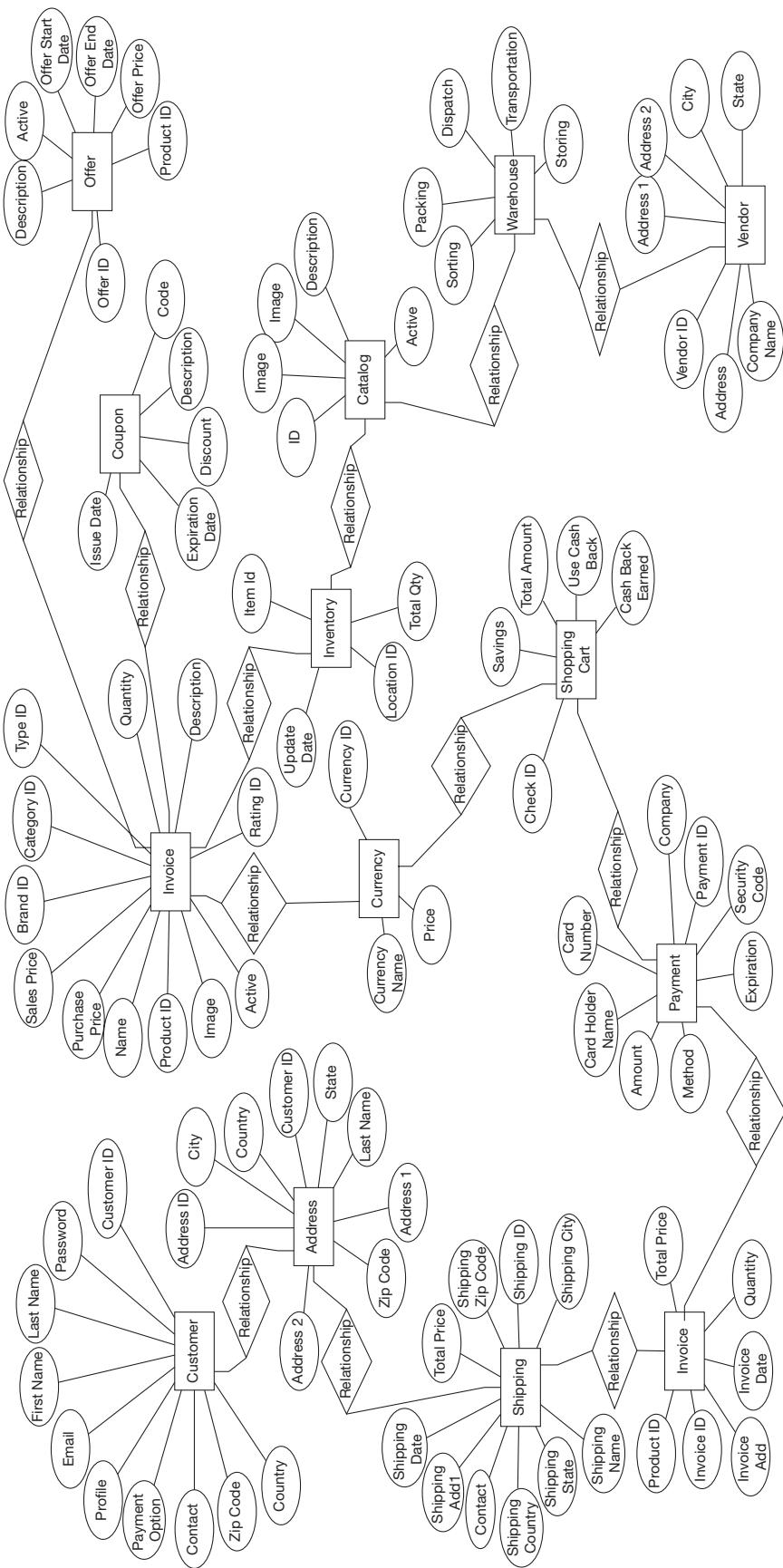


Figure 7.1 Entity relationship diagram of e-commerce entities.

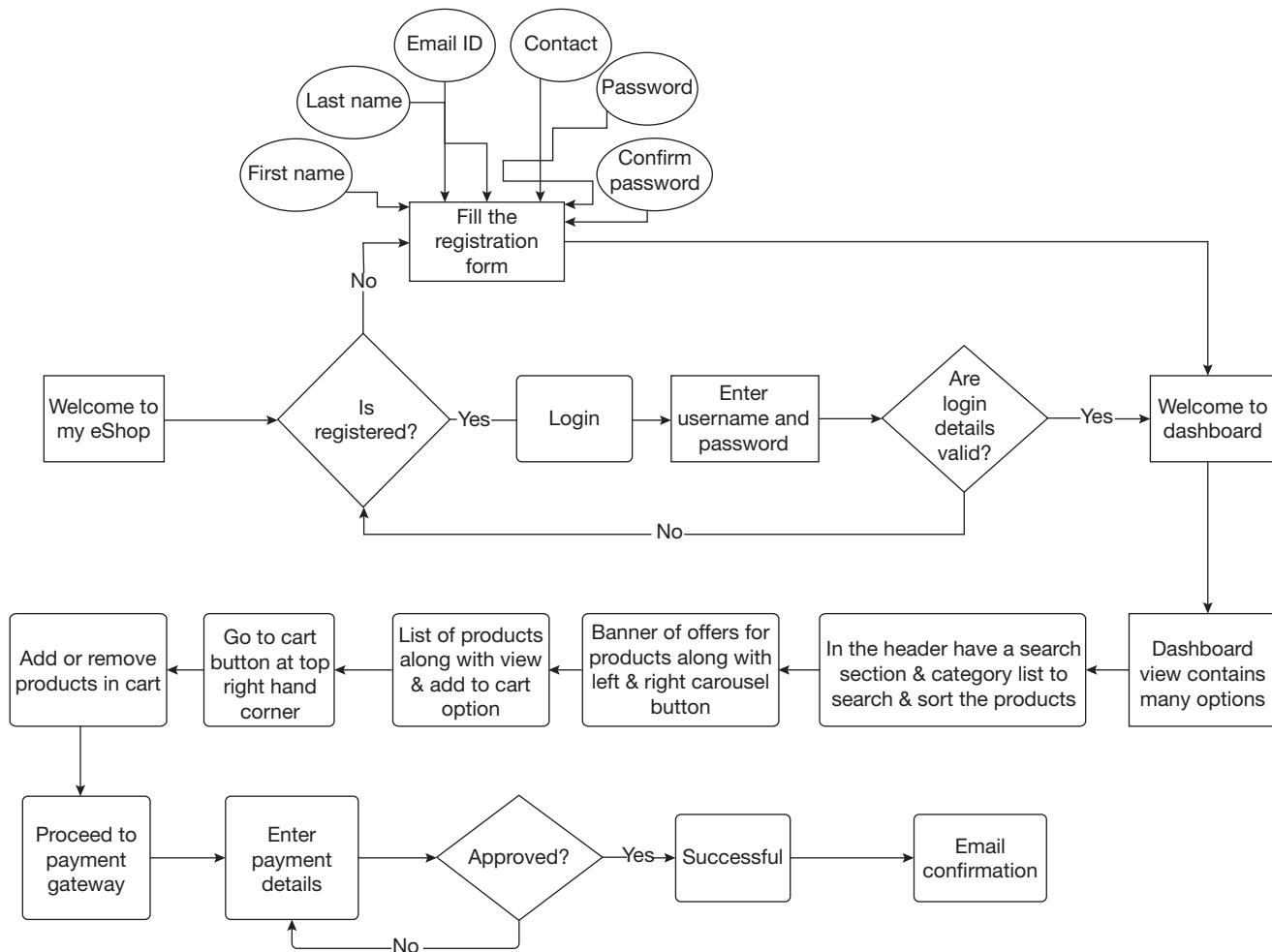


Figure 7.2 Login, registration, search, purchase, and payment flowchart.

7.3.1 Home Page

After saving the file, let us continue working on the page "*home.html*". We now need to add various HTML elements to make it functional.

First, we will start with standard HTML structure that we have learned in Chapter 3.

```

<!DOCTYPE html>
<html>
    <head>
        <title>My First HTML Example</title>
    </head>
    <body>
        <h1>Lets Learn HTML History</h1>
        <p>HTML is the most popular language for webpages that was created in early
        1990s</p>
    </body>
</html>

```

Now let us replace the text with our required inputs.

```
<!DOCTYPE html>
<html>
  <head>
    <title>MyEShop</title>
  </head>
  <body>
    <h1>Welcome to MyEShop</h1>
    <p>Buy Any Product you Like at a Reasonable Price</p>
  </body>
</html>
```

If you find difficulty understanding the above code, you can refer Chapter 3 where we have explained this in detail.

Now let us expand this further by adding one element type at a time.

7.3.2 Header

In this section we will create a header of the website that contains various elements such as logo, cart, navigation menu bar, and shopping cart.

7.3.3 Table

Let us add a table structure using the following code:

```
<table><tr><td></td></tr></table>
```

The explanation of the above elements is as follows:

<code><table></code>	HTML table element
<code><tr></code>	Table row
<code><td></code>	Table data/cell
<code></table></code>	Table closing tag
<code></tr></code>	Table row closing tag
<code></td></code>	Table cell closing tag
<code><th></code>	Table header tag
<code></th></code>	Table header closing tag

7.3.4 Logo

Let us add a logo image. We can use any geometric shape, preferably a circle, a square, or a rectangle.

```

```

<code></code>	Images are defined; contains only attributes and does not have a closing tag.
<code>Src</code>	Specifies the URL of the image.
<code>Alt</code>	Provides an alternate text for an image. If image does not get rendered, the text in alt tag is shown instead.
<code>Height</code>	Defines height of the image in pixels.
<code>Width</code>	Defines width of the image in pixels.

7.3.5 Search Bar

Let us add an input tab for search bar; we use a rectangular geometric shape.

```
<form><input type="text" placeholder="Search.."></form>
```

<i><form></i>	Defines a form that is used to collect user input.
<i><input></i>	Takes input type element.
<i><input type= "text"</i>	Allows text input.
<i>Placeholder</i>	Gives provision to show a short descriptive text.
<i></form></i>	Form closing tag.

7.3.6 Category List

Create a tab dedicated for listing categories of products. It should allow single selection. However, if you like to add multiple selection option, you just need to add “multiple” word to `<select>` tag like `<select multiple>`.

```
<select>
  <option>All</option>
  <option>CLOTHES </option>
  <option>FOOD AND BEVERAGES </option>
  <option>HEALTH & BEAUTY </option>
  <option>SPORTS & LEISURE </option>
  <option>BOOKS & ENTERTAINMENTS </option>
</select>
```

<i><select></i>	Opening tag to create a drop-down list.
<i><option></i>	Defines available options in the list.
<i></option></i>	Closing option tag.
<i></select></i>	Select closing tag.

7.3.7 Search Button

Create a search button that will be used to search user-entered text. It will be a simple button with title. Later, you can add action to this button so search can be performed on the entered text from the search bar.

```
<button>Search</button>
```

<i><button></i>	Defines a clickable button.
<i></button></i>	Button closing tag.

7.3.8 Shopping Cart

Create a dedicated box to show a shopping cart. This cart will contain total number of items that user has added while shopping on the site. In addition, this box will also show the total price for all the products that are in the cart.

```
<span><i> [ 0 ] Items in your cart </i></span>
```

	Useful to group in-line elements in a document.
<i>	Displays a text in italic.
</i>	Italic closing tag.
	Span closing tag.

7.3.9 Navigation Menu – Home

Create a home button for the website.

```
<a href="home.html">Home</a>
```

<a>	Defines a clickable anchor link.
<i>href</i>	This attribute sets the resource URL of page which will load upon clicking on this anchor tag.
	Anchor link closing tag

7.3.9.1 Navigation Menu – Special Offers

Create a button dedicated to view special offers and discounts page

```
<a href="specialoffers.html">Special Offers</a>
```

7.3.9.2 Navigation Menu – Contact

Create a button dedicated to view contact information

```
<a href="contact.html">Contact</a>
```

7.3.9.3 Navigation Menu – Login

Create a button dedicated to open Login and sign up page.

```
<a href="login.html">Login</a>
```

7.3.10 Execution of Code

Now let us execute the final HTML code:

```

<!DOCTYPE html>
<html>
  <head>
    <title>MyEShop</title>
  </head>
  <body>
    <h1>Welcome to MyEShop</h1>
    <p>Buy Any Product you Like at a Reasonable Price</p>

    <table>
      <tr>
        <td></td>
        <td><form><input type="text" placeholder="Search.." /></form></td>
        <td>
          <select>
            <option>All</option>
            <option>CLOTHES </option>
            <option>FOOD AND BEVERAGES </option>
            <option>HEALTH & BEAUTY </option>
            <option>SPORTS & LEISURE </option>
            <option>BOOKS & ENTERTAINMENTS </option>
          </select>
        </td>
        <td><button>Search</button></td>
        <td><span><i> [ 0 ] Items in your cart </i></span></td>
        <td><a href="home.html">Home</a></td>
        <td><a href="specialoffers.html">Special Offers</a></td>
        <td><a href="contact.html">Contact</a></td>
        <td><a href="login.html">Login</a></td>
      </tr>
    </table>

  </body>
</html>

```

If you open this file in a browser, you will see the following result. Please note, we are not seeing any image in the place where we put logo, instead we see a text “*myeshop logo*”. If you look closely, you will notice that “*myeshop logo*” is the alt text we use on the `` tag. So, when it does not find the specified image, it shows the alt text instead.

Welcome to MyEShop

Buy Any Product you Like at a Reasonable Price

myeshop logo All [0] Items in your cart [Home](#) [Special Offers](#) [Contact](#) [Login](#)

Based on the above example, continue the development according to the page flow diagram shown in Figure 7.2. Add more HTML elements to this page and create more pages according to the Navigation Bar. Take help of Entity Relationship Diagram to identify these pages and how to structure them. You may also create page flow diagram before starting the coding activity.

**QUICK
CHALLENGE**

Tom has created his first HTML page and added all the elements we have discussed so far in this chapter. Now, he would like to add Marquee Tag, Checkbox, and compare button. Help him code these elements.



7.4 | Adding CSS to the HTML Page

Now we will design the page and make it more presentable by adding CSS elements to it (we have already learned this in Chapter 4). First, we will add background color to the page so our page looks a little visually appealing.

7.4.1 Background Color

To add a background color to the entire page, we need to add it to the root element `<body>` of the page. For this type of inline style, we need to use “style” attribute on `<body>` element and inside “style” attribute set the background color.

```
<body style="background-color:#A2E9FF;">
```

See the following screenshot that shows the changed background color. Also note that `#A2E9FF` is a Hex Color Code. You can use various online websites to find a color code that you like.

Welcome to MyEShop

Buy Any Product you Like at a Reasonable Price

myeshop logo All [0] Items in your cart [Home](#) [Special Offers](#) [Contact](#) [Login](#)

7.4.2 Heading: Center Align

We can also use CSS to align text. See the following example which shows how to align text using inline CSS.

```
<h1 style="text-align:center;">Welcome to MyEShop</h1>
```

This text styling can be applied on any element. See the below example of `<p>` tag.

```
<p style="text-align:center;">Buy Any Product you Like at a Reasonable Price</p>
```

Above changes will bring the following changes to the pages. Please note that we have used “`text-align:center;`” to make the text look center aligned. Similarly, you can make text left, right, or justify. For that you can use “`text-align:left;`”, “`text-align:right;`”, “`text-align:justify;`”.

Welcome to MyEShop

Buy Any Product you Like at a Reasonable Price

myeshop logo All [0] Items in your cart [Home](#) [Special Offers](#) [Contact](#) [Login](#)

7.4.3 Navigation Menu Color

Now, let us change navigation bar menu color and make it look little different. In this example, we will add more than one styling to the element. See the following code that will help us to make the link look like a button.

```
<a style="background-color: red; color: white; padding: 1em 1.5em; text-decoration: none; -text-transform: uppercase;" href="home.html">Home</a>
```

The above code produces the following result. See the change in look compared to the other links on which we did not apply this styling.



Let us understand the style elements that we used.

1. **background-color:** This element sets the background color.
2. **color:** This element sets color for text.
3. **padding:** This element adds space on all four directions like left, right, top, and bottom.
4. **text-decoration:** This element dictates how anchor tag should be decorated.
5. **text-transform:** This element changes case for the text.

Now you can see how easy it is to change look and feel of a page with a very simple code.

You might have noticed the main title and subtitle are aligned center however the entire row which contains the logo, search bar, category dropdown, search button and navigation menu looks like left aligned. You can fix this by wrapping the `<table>` element in a div and setting its position to center like the following:

```
<div align="center"><table>....</table></div>
```

Repeat this process for all other anchor links and if needed change the color of your choice.

7.4.4 Fix the Image Problem

You might have noticed another problem like our `<table>` alignment and that is the `` problem. Our logo is not shown, instead it is showing the alt attribute we have used on the image element. It is very easy to fix the image problem we are experiencing for logo. As we have discussed earlier, this problem is there because browser does not find the mentioned image in the same directory as `home.html` file. Hence, to fix this problem, add the logo image named "`myeshoplogo.png`" to the same directory where "`home.html`" is placed.

See the following image which shows the image is clearly visible and alt text is hidden because browser can now find the image to display.



Finally, the code looks as follows:

```
<!DOCTYPE html>
<html>
    <head>
        <title>MyeShop</title>
    </head>
    <body style="background-color:#A2E9FF;">
        <h1 style="text-align:center;">Welcome to MyeShop</h1>
        <p style="text-align:center;">Buy Any Product you Like at a Reasonable Price</p>

        <table>
            <tr>
                <td></td>
                <td><form><input type="text" placeholder="Search.." /></form></td>
                <td>
                    <select>
                        <option>All</option>
                        <option>CLOTHES </option>
                        <option>FOOD AND BEVERAGES </option>
                        <option>HEALTH & BEAUTY </option>
                        <option>SPORTS & LEISURE </option>
                        <option>BOOKS & ENTERTAINMENTS </option>
                    </select>
                </td>
                <td><button>Search</button></td>
                <td><span><i> [ 0 ] Items in your cart </i></span></td>
                <td><a style="background-color: red; color: white; padding: 1em 1.5em; -text-decoration: none; text-transform: uppercase;" href="home.html">Home</a></td>
                <td><a href="specialoffers.html">Special Offers</a></td>
                <td><a href="contact.html">Contact</a></td>
                <td><a href="login.html">Login</a></td>
            </tr>
        </table>

    </body>
</html>
```

Summary

We have learned how to read the architectural diagrams we have created in Chapter 2. The aim of this chapter is not to give you a finished web application but to give you a starting point to build your own.

In this chapter, we have learned the following concepts:

1. Reading architectural diagrams.
2. Understand Page Flow diagram and think of creating pages.
3. Start with boiler plate code and add various elements to get desired results.
4. Use of common HTML elements.
5. Add CSS to HTML elements to make them look good.

In Chapter 8, we will take one step further and add jQuery to this page to add some programming logic.

Multiple Choices Questions

1. Which of the following tags is used to indicate the beginning of a paragraph?
 - (a) <TR>
 - (b) <p>
 - (c)

 - (d) <TD>
2. Markup tags are used to inform the web browser about which of the following?
 - (a) How to display the page.
 - (b) How to display a message box on the page.
 - (c) How to organize the page.
 - (d) None of the above
3. The file of map definition is usually stored in which of the following?
 - (a) RECYCLE-bin
 - (b) BIN
4. Which of the following CSS properties defines the bottom-left corner shape of the border?
 - (a) border-corner-radius
 - (b) border-radius
 - (c) border-left-radius
 - (d) border-bottom-left-radius
5. Which of the following CSS properties can be used for collapsing the borders between table cells?
 - (a) collapse-border
 - (b) border-collapse
 - (c) border-cell
 - (d) border

Review Questions

1. How do we use ERD to decide on creating HTML pages?
2. How can you add CSS to the HTML code?
3. How do you fix the image problem where the text given in the alt attribute is shown?
4. How do you align table element to center?
5. How do you create a multiple selection dropdown?

Exercise

Continue developing this further and add more and more styling to this page to make it attractive. Also add more pages

and styling to those. In the end of the exercise, you will have a nice-looking ecommerce front design.

Project Idea

Take the example of a school management system in which school admins can add, update, and delete students. Teachers can view students' data and also grade students on class tests.

Design front end pages in HTML and style them with CSS for the school management system.

Recommended Readings

1. Jon Duckett. 2011. *HTML and CSS: Design and Build Websites*. Wiley: New Delhi
2. Laura Lemay, Rafe Colburn, Jennifer Kyrnin .2016. *Mastering HTML, CSS & Javascript Web Publishing*. BPB Publications
3. Paul McFedries. 2018. *Web Coding & Development All-in-One For Dummies*. Wiley: New Delhi

Use of jQuery on HTML CSS

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- How to add jQuery to HTML pages.
- Various jQuery functions.
- Development with jQuery.
- How to use any text editor to code jQuery.

8.1 | Getting Started with jQuery



For jQuery you do not really need to set up any environment. As is the case with HTML, you can use any text editor you like. You can use the same editor you have used for coding HTML and CSS as well.

The only thing you need to make sure is that you add jQuery library to the page. There are two ways of doing this: First way is to add it from external sources like CDN or Google CDN or jQuery's own website or your own server. It really does not matter. The second way is to download jQuery library and keep it in the same place as "*home.html*". If you change the place, you just need to give the proper path of that location. For example, if some developer likes to keep this file in a folder named "js", your path will look like "js/jquerylibrary.js". In this case, *jquerylibrary.js* is the name of the current version of the library. The name can be anything you like as you are keeping it on your server. However, if you are using a hosted version from CDN then you cannot change the filename. Let us see the following example of using a hosted version of jQuery file.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
```

This is how you get jQuery file from a CDN. In our case, we have used Google CDN. As you already learned, this `<script>` tag will go under `<head>` tag.

Now, let us use our home page and start adding jQuery code to it.

8.2 | Home Page with jQuery



To start with let us add functionality to the search button we have added on the page. This button is intended to provide the list of all the products in the system which match the search criteria given by user in the search textbox.

In order to access the button in jQuery code, we need to add an "*id*" attribute to the `<button>` tag so that we can access it. Let us add "*id*" attribute first:

```
<button id="searchbtn">Search</button>
```

Now, we need to add jQuery code to access this button and capture the click action. So, when a user clicks on it, we can execute the code we want. Now, let us start building the jQuery code for the same. We will add another `<script></script>` block to add our jQuery code.

As you have learned in Chapter 5, we can place our code inside `$(document).ready(function() {})` to make sure we have all the HTML elements loaded properly before accessing them.

After doing this, our `<head>` section will look as follows:

```
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js" >
</script>
    <script>
        $( document ).ready(function() {
            });
    </script>
    <title>MyEShop</title>
</head>
```

If you are not sure about the jQuery syntax we have used, please visit Chapter 5 again to clear your doubts on jQuery. Now, let us write the code to access button with *id* selector.

```
<script>
    $( document ).ready(function() {
        $("#searchbtn").click(function() {
            });
    });
</script>
```

As you have learned, id selection is captured by using “`#`” and then the text mentioned in the “*id*” attribute. In our search button case, the id of the search button is “`searchbtn`” and hence we have used “`#searchbtn`” to select this particular button. Then we need to capture the click event so we have added “`.click()`” to capture this event. However, we need to place our code in between that block so our code will go inside “`$("#searchbtn").click(function() {});`”. Let us move on to add an alert box to make sure the click event is properly capturing the event.

This is how our code looks like for the alert box.

```
<script>
    $( document ).ready(function() {
        $("#searchbtn").click(function() {
            alert("Search button is clicked");
        });
    });
</script>
```

Now, let us save an and open this file in a browser window; we will see the following result.



Once the page is loaded, a pop-up box is visible which shows a message. This appears when we clicked on the “Search” button. Now we know our click event is working.

In order to make a proper search and display function, we need to consider the text from the search text box and text from the category dropdown. We will use these two values to filter the result from the database to get the selected products back. So, let us code to capture these two values.

First read the textbox value. For this, we have to do the same thing we did for the button. We need to add the “*id*” attribute to the textbox so we can use *id* selector in jQuery to get access to this particular textbox.

This is how our code looks like after adding the id attribute.

```
<input id="searchtxt" type="text" placeholder="Search..">
```

Now let us add code on the jQuery side to access this and capture the value from it. In order to get the value from this textbox, we need to use “*.val()*” function. This is how our code will look after adding it.

```
<script>
    $( document ).ready(function() {
        $("#searchbtn").click(function() {
            alert("Search button is clicked. Value of Search Text is" +
        $("#searchtxt").val());
        });
    });
</script>
```

For now, we are just adding it in the alert box so we can see the captured value in the popup. Now let us run this in a browser window.



As you can see in the screenshot, we can get the captured value from the textbox. Since we can also filter from the category item, we need to capture the selected category as well. For this we will repeat the id selector procedure.

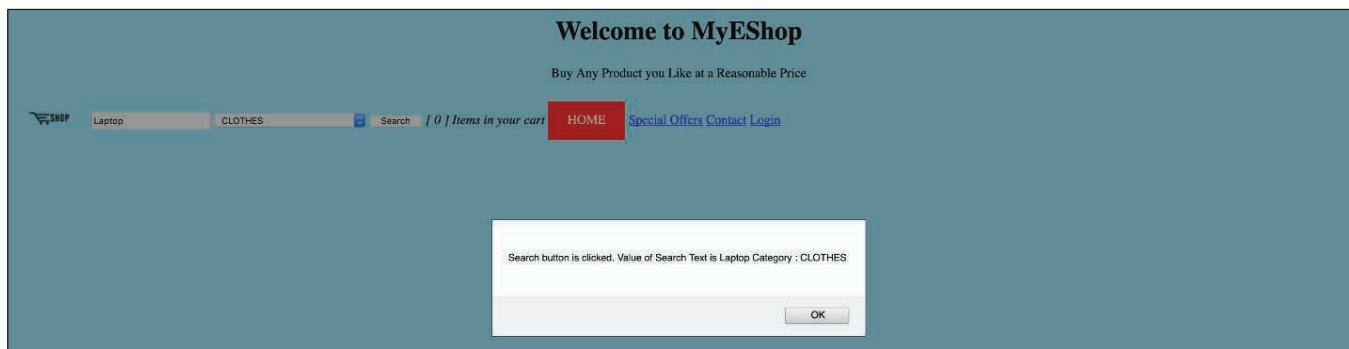
This is what our code looks like when we add id to the select element.

```
<select id="categorysel">
    <option>All</option>
    <option>CLOTHES </option>
    <option>FOOD AND BEVERAGES </option>
    <option>HEALTH & BEAUTY </option>
    <option>SPORTS & LEISURE </option>
    <option>BOOKS & ENTERTAINMENTS </option>
</select>
```

Now, let us use *id* selector on jQuery side to access this select element. To capture selected option, we can use the following code. Please note that here we are not using “*.val()*” but using “*.html()*”. “*.html()*” is used to get data from the element as elements do not have value but only content data. On the other hand, input box has value.

```
<script>
    $( document ).ready(function() {
        $( "#searchbtn" ).click(function() {
            alert("Search button is clicked. Value of Search Text is" +
$( "#searchtxt" ).val() + " Category : " + $( "#categorysel" ).children(":selected") .
html());
        });
    });
</script>
```

Also, note that to get a value of selected option, we are using “*(“:selected”)*” on all the children of the category dropdown. jQuery traverses all the children and identifies the selected one. After that, as we just discussed, we get “*html()*” to fetch value from the element. See the following screenshot:



This is good start for you to continue your journey into developing this application further. You can add any action you like in place of alert statement. However, to make it functional we will be calling web services from this code block and parsing the result we are going to get from the webserver. We will work on this ajax call code in Chapter 24, once our web service is ready in Chapters 22 and 23.

In the end, this is how our complete file looks.

```

<!DOCTYPE html>
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
        <script>
            $( document ).ready(function() {
                $( "#searchbtn" ).click(function() {
                    alert("Search button is clicked. Value of Search Text is " +
$($("#searchtxt").val() + " Category : " + $("#categorysel").children(":selected") .
html()));
                });
            });
        </script>
        <title>MyEShop</title>
    </head>
    <body style="background-color:#A2E9FF;">
        <h1 style="text-align:center;">Welcome to MyEShop</h1>
        <p style="text-align:center;">Buy Any Product you Like at a Reasonable Price</p>

        <table>
            <tr>
                <td></td>
                <td><form> <input id="searchtxt" type="text" placeholder="Search..">
</form></td>
                <td>
                    <select id="categorysel">
                        <option>All</option>
                        <option>CLOTHES </option>
                        <option>FOOD AND BEVERAGES </option>
                        <option>HEALTH & BEAUTY </option>
                        <option>SPORTS & LEISURE </option>
                        <option>BOOKS & ENTERTAINMENTS </option>
                    </select>
                </td>
                <td><button id="searchbtn">Search</button></td>
                <td><span ><i> [ 0 ] Items in your cart </i> </span></td>
                <td><a style="background-color: red;color: white;padding: 1em 1.5em;-
text-decoration: none;text-transform: uppercase;" href="home.html">Home</a></td>
                <td><a href="specialoffers.html">Special Offers</a></td>
                <td><a href="contact.html">Contact</a></td>
                <td><a href="login.html">Login</a></td>
            </tr>
        </table>
    </body>
</html>

```

**QUICK
CHALLENGE**

Tom was able to design an HTML page with CSS and add jQuery code to change page content based on selected value from a dropdown. Help him write the code to add ajax calls to fetch the latest product reviews from Google or other websites. Also, he was thinking of adding a box to show all the Facebook posts related to the product from his shop's Facebook page. Help him figure out the Facebook integration.

Summary

As you have seen, it is very easy to add jQuery to any page and use it without installing anything on the server or client side. The only thing we need to do is get the jQuery library. jQuery code can be written in any simple text editor as well. So, getting started with jQuery is a simple process.

In this chapter, we have learned the following concepts:

1. Process for accessing jQuery library.
2. Adding jQuery library to HTML pages.
3. Various jQuery functions that we can use on the page.
4. Using id selector to get a particular element to work on.

In Chapter 9, we will use Bootstrap to add some responsive behavior to our pages, so that they render properly in any type of device with varying screen sizes.

Multiple Choices Questions

1. Which of the following options is used to alternately expand and collapse a page element?
 - (a) .toggle()
 - (b) .trigger()
 - (c) .stopPropagation()
 - (d) .hover()
2. Which of the following options is utilized to return/set arbitrary data to/from an element?
 - (a) .item()
 - (b) .data()
 - (c) .all
 - (d) None of the above
3. JQuery is a W3C standard.
- (a) True
- (b) False
4. Which of the following JQuery methods is used to hide selected elements?
 - (a) hide()
 - (b) display:none)
 - (c) hidden()
 - (d) visible(false)
5. Which scripting language is JQuery written in?
 - (a) C++
 - (b) JavaScript
 - (c) C#
 - (d) VBScript

Review Questions

1. How do you get the selected values from a dropdown?
2. What are the advantages of using jQuery from a CDN server over using a local downloaded copy on the server?
3. What are the disadvantages of using jQuery from a CDN server over using a local downloaded copy on the server?
4. How do we capture a click event for a button?
5. How do we use an alert popup to notify user?

Exercise

Carry on with the development of this page. Think of programming logic you can add on this page to improve this further. Also add jQuery on the new pages you have created in the exercise in Chapter 7.

Project Idea

Take the example of the school management system from Chapter 7's Project Idea section. Add interactivity to the HTML and CSS pages you have created for the requirements mentioned in that section. Make sure you add validation to the new student form.

Recommended Readings

1. David Sawyer McFarland. 2014. *JavaScript & jQuery: The Missing Manual*. O'Reilly: Massachusetts
2. John Pollock. 2014. *jQuery: A Beginner's Guide*. McGraw-Hill Education: New York
3. Jon Duckett. 2014. *Web Design with HTML, CSS, JavaScript and jQuery Set*. Wiley: New Delhi

Use of Bootstrap to Make HTML Responsive

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- How to add Bootstrap to HTML pages to make them responsive.
- How to use various Bootstrap functions.
- Hands-on development with Bootstrap.
- How to use any text editor to code Bootstrap.

9.1 | Setting up Environment



For Bootstrap, you do not need to set up any environment. As is the case with HTML, you can use any text editor you like. You can use the same editor you had used for coding HTML, CSS, and jQuery as well. The only thing you need to make sure is that you add Bootstrap library to the page. Just like jQuery, there are two ways to do this. The first way is to add it from the external sources like CDN or Google, CDN or Bootstrap's website or your own server, it really does not matter. The second way is to download Bootstrap library and keep it in the same place as "*home.html*". If you change the place, you just need to give the proper path of that location. For example, some developers like to keep this file in a folder named "*bootstrap*". So, your path will be "*bootstrap/bootstrap.js*". In this case, *bootstrap.js* is the name of the current version of the library. The name can be anything you like as you are keeping it on your server. However, if you are using a hosted version from CDN, then you cannot change the filename. Let us see the following example:

```
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-JJSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
```

We need one more *js* to be included before *bootstrap.js*. This library called *popper.js*. The link is as follows:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.
js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzwj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
```

For Bootstrap we need to add Bootstrap CSS as well. The procedure is exactly the same as mentioned above. There are two more attributes you can see here. First one is "*integrity*" which helps verify the authenticity of the file and second is "*crossorigin*" which is needed to tell the browser that cross-origin request is ok. This is an important concept if we are trying to access a file

from a different domain than ours. Almost all the newer browsers block the request thinking that this is **cross-site scripting (XSS)** attack. The following code shows how to get Bootstrap CSS file from a CDN server.

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/
/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY
/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
```

As you may have noticed in the CSS case, we are not using `<script>` tag but instead are using the `<link>` tag. In the attribute we are telling the browser that this is a stylesheet by mentioning it in the “`rel`” attribute. Then in the “`href`” tag, we are mentioning the path. Other two attributes we have already covered.

Now you have seen how we got both the Bootstrap files from a CDN. In our case, we have used Bootstrap CDN. As you already learnt, the `<script>` tag will go under `<head>` tag; similarly, `<link>` will go under `<head>` tag too.

Since we are adding Bootstrap to make this application responsive, we need to tell the browser about it. This is something we can do by adding a `<meta>` tag under `<head>` element:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

In the above line, by adding “`width=device-width`”, we are telling the browser to set the width of the page as per the screen width. By “`initial-scale=1`”, we are telling the browser to set the initial zoom level to 1 when the page is first loaded.

After all of the above, our header look as follows:

```
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js" >
</script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.
min.js" integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz
0W1" crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.
js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFF/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
    <script>
        $( document ).ready(function() {
            $( "#searchbtn" ).click(function() {
                alert("Search button is clicked. Value of Search Text is " +
                $("#searchtxt").val() + " Category : " + $("#categorysel").children(":selected") .
                html());
            });
        });
    </script>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/
/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY
/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>MyEShop</title>
</head>
```

In the above code, we are using the following jQuery code which captures the click event on the search button we have added.

```
$( "#searchbtn" ).click(function() {
    alert("Search button is clicked. Value of Search Text is " +
    $("#searchtxt").val() + " Category : " + $("#categorysel").children(":selected") .
    html());
});
```

Now let's use our Home page and start adding Bootstrap code to it.

9.1.1 Home Page with Bootstrap



First let us add the container as we have learnt in Chapter 6. We will add this with div inside `<body>` tag:

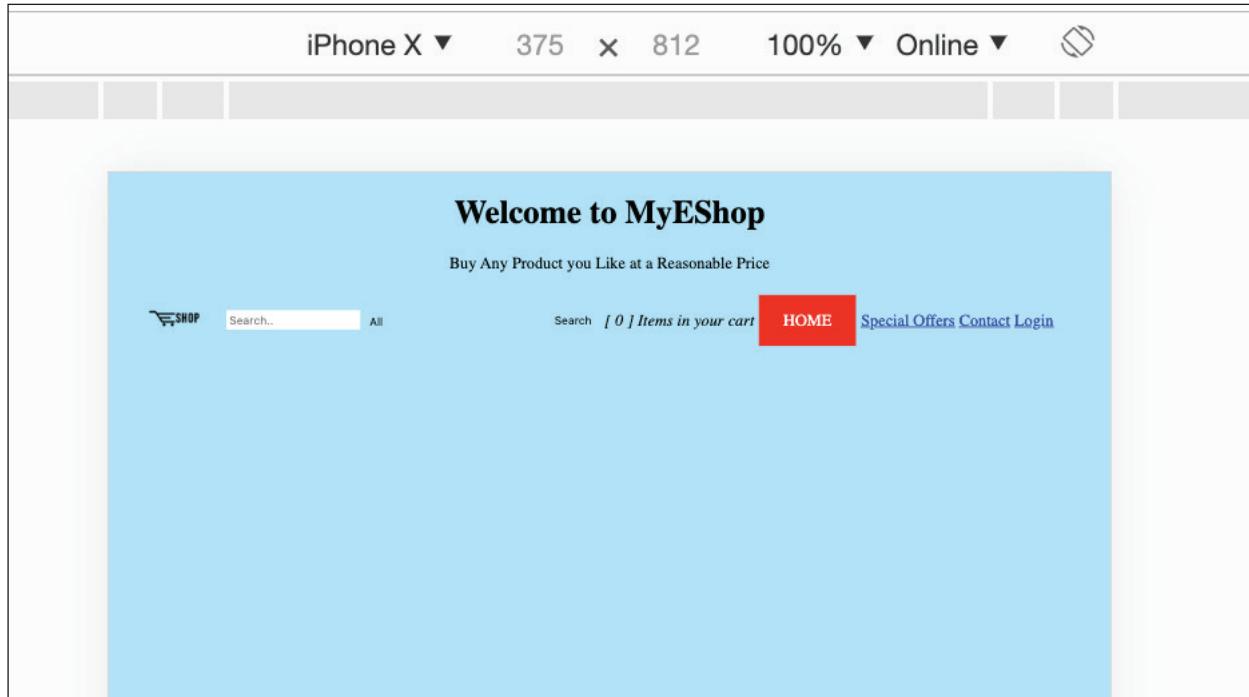
```
<body style="background-color:#A2E9FF;">
    <div class="container">
```

This is how we tell Bootstrap which type of container we want. For now, let us go ahead with this.

QUICK CHALLENGE

Try using `container-fluid` and see how the page gets rendered.

One thing you should remember is that `<table>` is not responsive and will not render properly on mobile devices. See the following screenshot indicating how the page looks on iPhone X.



This is not how we want the page to look because the alignment is misplaced a bit. Hence, we need to use `<div>` and let Bootstrap arrange this for us. Hence, we must replace the following code with Bootstrap Grid:

```

<table>
    <tr>
        <td></td>
        <td><form> <input id="searchtxt" type="text" placeholder="Search..">
</form></td>
        <td>
            <select id="categorysel">
                <option>All</option>
                <option>CLOTHES </option>
                <option>FOOD AND BEVERAGES </option>
                <option>HEALTH & BEAUTY </option>
                <option>SPORTS & LEISURE </option>
                <option>BOOKS & ENTERTAINMENTS </option>
            </select>
        </td>
        <td><button id="searchbtn">Search</button></td>
        <td><span ><i> [ 0 ] Items in your cart </i> </span></td>
        <td><a style="background-color: red;color: white;padding: 1em 1.5em;-text-decoration: none;text-transform: uppercase;" href="home.html">Home</a></td>
        <td><a href="specialoffers.html">Special Offers</a></td>
        <td><a href="contact.html">Contact</a></td>
        <td><a href="login.html">Login</a></td>
    </tr>
</table>

```

The above code gets replaced with the following code:

```

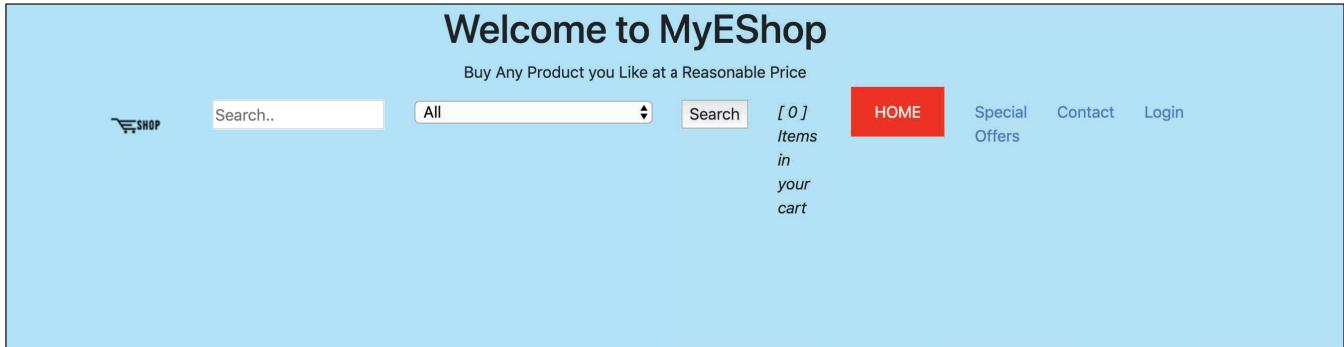
<div class="row">

    <div class="col"></div>
    <div class="col"><form> <input id="searchtxt" type="text"
placeholder="Search.."></form></div>
    <div class="col">
        <select id="categorysel">
            <option>All</option>
            <option>CLOTHES </option>
            <option>FOOD AND BEVERAGES </option>
            <option>HEALTH & BEAUTY </option>
            <option>SPORTS & LEISURE </option>
            <option>BOOKS & ENTERTAINMENTS </option>
        </select>
    </div>
    <div class="col"><button id="searchbtn">Search</button></div>
    <div class="col"><span ><i> [ 0 ] Items in your cart </i> </span></div>
    <div class="col"><a style="background-color: red;color: white;padding: 1em 1.5em;text-decoration: none;text-transform: uppercase;" href="home.html">Home</a></div>
    <div class="col"><a href="specialoffers.html">Special Offers</a></div>
    <div class="col"><a href="contact.html">Contact</a></div>
    <div class="col"><a href="login.html">Login</a></div>
</div>

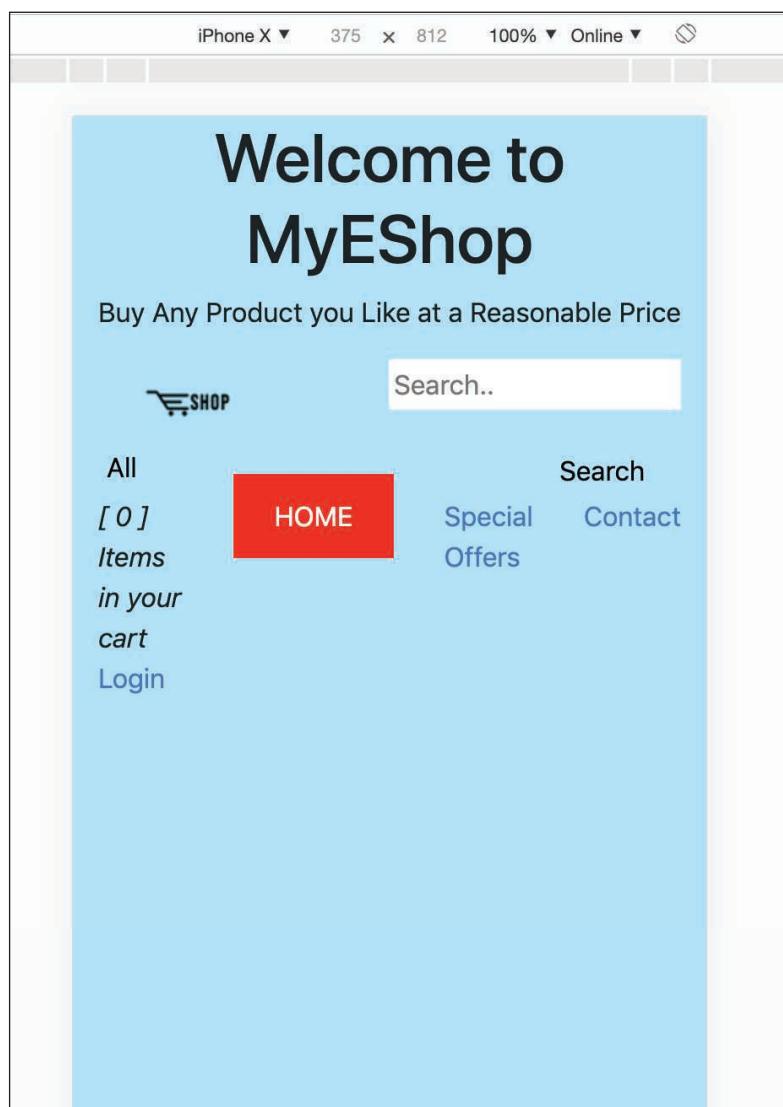
```

When we add bootstrap Grid like `<div class="col">` in place of `<table>`, the page looks good on both the devices – laptop as well as iPhone X (see both the screenshots below):

On Laptop:



On iPhone X:



Much better, right? That is the beauty of using Bootstrap. After adding this code, our final code looks as follows:

```

<!DOCTYPE html>
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"
        ></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-UO2eT0CpHqdsJQ6hJty5KVphPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-J-SvZiR4YqE4KoQcH2hZqy+jcB7PzU9y6ZDxQZTtq+XG4rPfLXwZuNzLXwq6WZlJRM" crossorigin="anonymous"></script>
        <script>
            $( document ).ready(function() {
                $( "#searchbtn" ).click(function() {
                    alert("Search button is clicked. Value of Search Text is " +
                    $("#searchtxt").val() + " Category : " + $("#categorysel").children(":selected") .
                    html());
                });
            });
        </script>
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>MyEShop</title>
    </head>
    <body style="background-color:#A2E9FF;">
        <div class="container">
            <h1 style="text-align:center;">Welcome to MyEShop</h1>
            <p style="text-align:center;">Buy Any Product you Like at a Reasonable Price
        </p>

        <div class="row">
            <div class="col"></div>
            <div class="col"><form> <input id="searchtxt" type="text" placeholder="Search.."/></form></div>
            <div class="col">
                <select id="categorysel">
                    <option>All</option>
                    <option>CLOTHES </option>
                    <option>FOOD AND BEVERAGES </option>
                    <option>HEALTH & BEAUTY </option>
                    <option>SPORTS & LEISURE </option>
                    <option>BOOKS & ENTERTAINMENTS </option>
                </select>
            </div>
            <div class="col"><button id="searchbtn">Search</button></div>
            <div class="col"><span ><i> [ 0 ] Items in your cart </i> </span></div>
            <div class="col"><a style="background-color: red;color: white;padding: 1em 1.5em;text-decoration: none;text-transform: uppercase;" href="home.html">Home</a>
            </div>
            <div class="col"><a href="specialoffers.html">Special Offers</a></div>
            <div class="col"><a href="contact.html">Contact</a></div>
            <div class="col"><a href="login.html">Login</a></div>
        </div>
    </div>
</body>
</html>

```

Summary

As you have seen, it is very easy to add Bootstrap to any page and use it without installing anything on the server or client side. The only thing we need to do is get the jQuery, popper.js, Bootstrap JS, and Bootstrap CSS library. Bootstrap code can be written in any simple text editor as well. So getting started with Bootstrap is not at all a difficult task. Pages build with Bootstrap looks good on mobile devices.

In this chapter, we have learned the following concepts:

1. Accessing Bootstrap library and setting up the dependencies for it.
2. Adding Bootstrap library to HTML pages to use the responsive elements.
3. Various Bootstrap functions that we can use on the page.
4. Using grid in place of `<table>` to make it responsive on mobile devices.

In Chapter 10, we will learn about Java and how to use it to develop backend code. We will go through Java 9 and Java 11 to see the latest improvements. We will spend time to understand concepts like JVM, compiler, garbage collector, JRE, and JDK. Also we will briefly touch upon object-oriented programming and learn concepts like polymorphism, inheritance, abstraction, and encapsulation. We will also learn Java reserved words, classes and objects, variables, methods, constructors, etc.

Multiple-Choice Questions

1. How many columns are there in the Bootstrap grid system?
 (a) 12
 (b) 6
 (c) 14
 (d) 9
2. Which of the given plugins is used to cycle through elements, like a slideshow?
 (a) Scrollspy
 (b) Carousel
 (c) Slideshow
 (d) Orbit
3. What is the default size of H5 Bootstrap heading?
 (a) 16px
4. Which class creates a list of items?
 (a) list-group
 (b) menu-group
 (c) list-grp
 (d) lst-group
5. Which of the given class specifies a dangerous or potentially negative action?
 (a) .active
 (b) .warning
 (c) .danger
 (d) .success

Review Questions

1. What is the use of `crossorigin="anonymous"`?
2. What is the use of `popper.min.js`?
3. How do we create grid with Bootstrap?
4. What is the use of `<meta name="viewport" content="width=device-width, initial-scale=1">`?

Exercise

Carry on with the bootstrapping process of this page. Think of various areas that you can improve for mobile devices and add those elements. Also, add Bootstrap on the new pages you have added as mentioned in Chapter 7.

Project Idea

Take the example of a school management system from Chapter 8's Project Idea section. These pages now have a nice design and interactivity. Make sure they look nice on different screen sizes like iPad, mobile phones, etc. Add Bootstrap to these pages.

Recommended Readings

1. Jacob Lett. 2018. *Bootstrap Reference Guide: Quickly Reference All Classes and Common Code Snippets (Bootstrap 4 Tutorial Book 2)*. Bootstrap Creative
2. Jake Spurlock. 2013. *Bootstrap: Responsive Web Development*. O'Reilly: Massachusetts
3. Alan Forbes. 2014. *The Joy of Bootstrap: A smarter way to learn the world's most popular web framework*. Plum Island Publishing: Massachusetts

Introduction to Java Language

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Java 13, Java 11, and Java 9 improvements.
- Java concepts such as Java Virtual Machine, compiler, garbage collector, Java Runtime Environment, and Java Development Kit.
- Principles of object-oriented programming such as abstraction, inheritance, encapsulation, and polymorphism.
- How to install Java Development Kit.
- Importance of syntax and structure in Java.
- Reserved words, classes and objects, variables, methods, constructors, etc. using libraries.
- New features in Java 9 like Java Shell (RELP), Multi-Release JAR Files, Platform Support for Legacy Versions, Java Linker, Hash Algorithms, Modular Java Packaging, Tuning Improvements, G1GC, Process API Updates, and XML Catalogs.
- How to use the Integrated Development Environment such as Eclipse.

10.1 | Overview of Java



Java is one of the most popular languages used in the software world. It dominates most of the software – from household devices like microwave, washing machine, etc. to space technologies like NASA’s Mars Rovers. Java works on “Write Once, Run Anywhere” (WORA) principle and hence can transcend many platforms. You may write a Java program on any device and compile into a standard bytecode. This bytecode can then run inside a Java Virtual Machine (JVM) installed on any other device, even if it is running a different operating system. This gives tremendous benefit as your program is now portable. There are endless possibilities of what you can do with Java.

Java 13 is the latest version of the famous programming platform released on 17th September 2019 which brings 5 JDK Enhancement Proposals (JEPs) and 76 new core library elements. Majority of these core library elements are just simple additions to the `java.io` package. Following are the Features you can expect in Java 13. The details can be found on the openjdk website (<https://openjdk.java.net/projects/jdk/13/>)

JEP 350 Dynamic CDS Archives: Java 10 brought an interesting change “JEP 310: Application Class-Data Sharing” which improves startup performance and footprint by helping JVM reuse class-data archive. This CDS feature allows application classes to be included in the shared archive. This JEP 350 enhances this feature further where at the end of Java application execution it allows dynamic archiving of classes. This archive includes all loaded application classes and library classes other than the ones present in the default base-layer CDS archive.

JEP 351 ZGC: Uncommit Unused Memory: This JEP is related to the garbage collector for claiming the unused heap memory by improving the z garbage collector.

JEP 353 Reimplement the Legacy Socket API: This JEP intends to modernize `java.net.Socket` and `java.net.ServerSocket` APIs by replacing the underline implementation which will provide easy adaptation of user-mode threads also known as fibers. These old APIs are from JDK 1.0 which uses `PlainSocketImpl` for `SocketImpl`. This has been replaced by `NioSocketImpl` in Java 13.

JEP 354 Switch Expressions (Preview): This enhancement brings a new keyword ‘yield’ to return a value from the switch. Previous to Java 12, switch was using a variable and then break statement to return a value. See the following example:

```
String result = "";
switch (character) {
    case a, b:
        result = "This is a or b";
        break;
    case z:
        result = "This is z";
        break;
    default:
        result = "This is other than a, b, or z. ";
        break;
}
return result;
```

In Java 12, we can simply use break to return a value.

```
String result = switch (character) {
    case a, b:
        break "This is a or b";
    case z:
        break "This is z";
    default:
        break "This is other than a, b, or z. ";
};
return result;
```

In Java 13, Java 12’s break for returning value does not compile. Instead, we should use yield. See the following example:

```
return switch (character) {
    case a, b:
        yield "This is a or b";
    case z:
        yield "This is z";
    default:
        yield "This is other than a, b, or z. ";
};
```

However, the Java 12's introduction of new lambda-style syntax with the arrow is supported in Java 13. See the following example:

```
return switch (character) {
    case a, b -> "This is a or b";
    case z -> "This is z";
    default -> "This is other than a, b, or z. ";
};
```

JEP 355 Text Blocks (Preview): This JEP finally brings the most desired multi-line string literal, a text block that will allow to express strings which are placed on several lines. In the era of web development, many times it is essentials to send a multi-line data to the program but in the absence of such provision to accept, previous to Java 13 versions, we have to use multi-line as follows:

```
String myTextBlock = "<html>\n" +
    " <body>\n" +
        " <div>Hello World!!!</div>\n" +
    " </body>\n" +
"</html>\n";
```

But with Java 13's text block, we can simplify this as follows:

```
String myTextBlock = """
<html>
    <body>
        <div>Hello World!!!</div>
    </body>
</html>
""";
```

With this you can see, how Java is constantly striving to be the best programming languages in the world by modernizing many of its features and providing greater support to cloud based applications.

Before Java 13, Java 11 was the version which caught the attention of many. The update (Java11) has brought some interesting features as well as removed some of the older technology features. Following is the list taken from the official site which shows the newly added and removed features (<http://openjdk.java.net/projects/jdk/11/>).

1. JEP 181: Nest-Based Access Control
2. JEP 309: Dynamic Class-File Constants
3. JEP 315: Improve Aarch64 Intrinsics
4. JEP 318: Epsilon: A No-Op Garbage Collector
5. JEP 320: Remove the Java EE and CORBA Modules
6. JEP 321: HTTP Client (Standard)
7. JEP 323: Local-Variable Syntax for Lambda Parameters
8. JEP 324: Key Agreement with Curve25519 and Curve448

9. JEP 327: Unicode 10
10. JEP 328: Flight Recorder
11. JEP 329: ChaCha20 and Poly1305 Cryptographic Algorithms
12. JEP 330: Launch Single-File Source-Code Programs
13. JEP 331: Low-Overhead Heap Profiling
14. JEP 332: Transport Layer Security (TLS) 1.3
15. JEP 333: ZGC: A Scalable Low-Latency Garbage Collector(Experimental)
16. JEP 335: Deprecate the Nashorn JavaScript Engine
17. JEP 336: Deprecate the Pack200 Tools and API

Before this update, Java 9 brought some of the major changes. It was finalized in 2011 and released in September 2017. Oracle described that they wanted to bring out a version that had a better default garbage collector, ability to deal with large heap sizes, and an improved JVM with a self-tuning capability. The elements of Project Jigsaw are also implemented in this version, which calls for a modular design for quick improvements and future enhancements. The final version that came out late September 2017 has most of the issues ironed out, and now it is time to start using the powers of Java 9.

There are several enhancement protocols such as JDK Enhancement Proposals (JEP) which are implemented in the Java 9 version. Some notable ones are the following:

1. **JEP 193:** It introduces the use of variable handles with the ability to create equivalent functionality in a variety of situations.
2. **JEP 222:** It implements a fully functional Java Shell which allows the use of a command prompt to run single line functions and statements.
3. **JEP 266:** Java 9 is big on concurrency and brings in the elements of reactive programming. This is introduced using a flow class with the capability of allowing the use of individual threads.
4. **JEP 268:** It introduces the use of XML Catalogs that allow the use of other languages to be implemented and integrated with the Java programming efforts.
5. **JEP 282:** It introduces the linking tool creator in Java. It can combine different modules and use the available dependencies to create customized processing solutions. It produces an executable file which includes the JVM structure required to run the Java program in any environment.
6. **JEP 295:** It describes the use of Ahead of Time (AOT) compilation, which reduces the load on the hosting processors and offers significant advantages in a variety of settings.
7. **JSR 376:** It describes the modularization concept which is introduced as the latest update through Project Jigsaw creating the platform module system for producing enhanced functionality.

10.2 | Basic Java Concepts

Java is one of the most widely used programming languages. It is an object-oriented language that uses a system of class hierarchies to create and manage the required functionality. The main advantage of Java is that it is designed to run on any system with the support of a virtual machine, in the form of JVM. The virtual machine creates a desirable environment for a Java program on all physical systems. It works by creating a bytecode that can be simply executed in an installed JVM environment.

With the latest updates, Java further enhances the main capacity of the language to offer a portable solution, where there is no need to recompile the program. The main syntax of the language is driven from C, especially the C++ platform. However, it is different because it focuses on creating solutions, rather than offering low-level functions, which are a trademark of the C language.

Now, we will discuss a few important concepts that you need to understand before starting on writing programs in Java.

In the Java world, you will frequently come across with words like JDK, JRE, and JVM as these are the core-essence of Java programming language. As a programmer, you may never need to go in depth of these concepts as you will be just using them to configure in Integrated Development Environment (IDE), but it is good to understand them properly in order to use Java to its fullest potential. So, let us first understand each concept and then see how they differ. Figure 10.1 shows how these components are attached together.



Figure 10.1 Components of Java language.

10.2.1 Java Virtual Machine



Java Virtual Machine (JVM) is the name given to an abstract computing structure that allows a computer with any specification to run a Java-based program. Each JVM is uniquely defined by three concepts:

1. *Specification* that documents how the JVM needs to be implemented through the computing resources. A single specification will suffice for all types of implementations.
2. *Implementation* is the next notion. It is a computer program that codes the requirements presented by the JVM specification.
3. The last part is an *instance*. This is simply the implementation program working on an available process to execute the Java bytecode generated by Java compilers.

JVM is what makes Java platform independent. Every OS has different JVM. However, the output generated after the bytecode is the same across all OS.

10.2.2 Compiler



Java code can be written in any text-based application. However, it only becomes a source code when you save a program on a relevant platform in the form of a .java file. This file needs to be compiled in order to become a program, which can be used as an executable version. The Java compiler is responsible for checking the code for compliance with the language rules, such as following the defined syntax.

Once source code is found to be compatible and without errors, the compiler converts the program into bytecode which results in a .class file. This file contains the instructions which are required for running the program on any JVM. The compiler is different because it does not produce the machine code for a particular CPU; instead, it generates instructions which can be run on any device with the installed JVM or Java Runtime Environment (JRE).

10.2.3 Java Runtime Environment



Java Runtime Environment (JRE) is a software suite which is installed on physical machines to enable the execution of Java programs. It is a package that combines a JVM with the class libraries that are required to run any defined bytecode in Java. Oracle provides its own JRE as a package in the form of HotSpot – their proprietary software package.

10.2.4 Java Development Kit



Java Development Kit (JDK) is a complete collection of all the tools that Oracle provides for Java. It provides a combination of programming, compiling, and execution elements which are required by Java programmers. You can use the OpenJDK environment, which is an open-source solution offered by Oracle. There are several companies that also offer their development kits since JDKs are freely available for use.

Programmers need access to a JDK, which may include a graphical user interface (GUI) as well to help implement convenience in programming activities. We especially look at Eclipse in this guide to carry out the ideal Java-based projects.



Which one will you choose to install on your development machine – JDK or JRE?

Please follow Table 10.1 to understand the differences between JDK and JRE, so you will know which one to use and when.

Table 10.1 Differences between JDK and JRE

JDK	JRE
Development kit	Implementation of JVM
Useful for compiling Java program	Useful for running Java program
Contains Compiler and Debugger	Does not contain Compiler and Debugger
Needs more disk space	Smaller than JDK

Figure 10.2 shows how your Java program gets executed through JDK, JVM, and JRE.

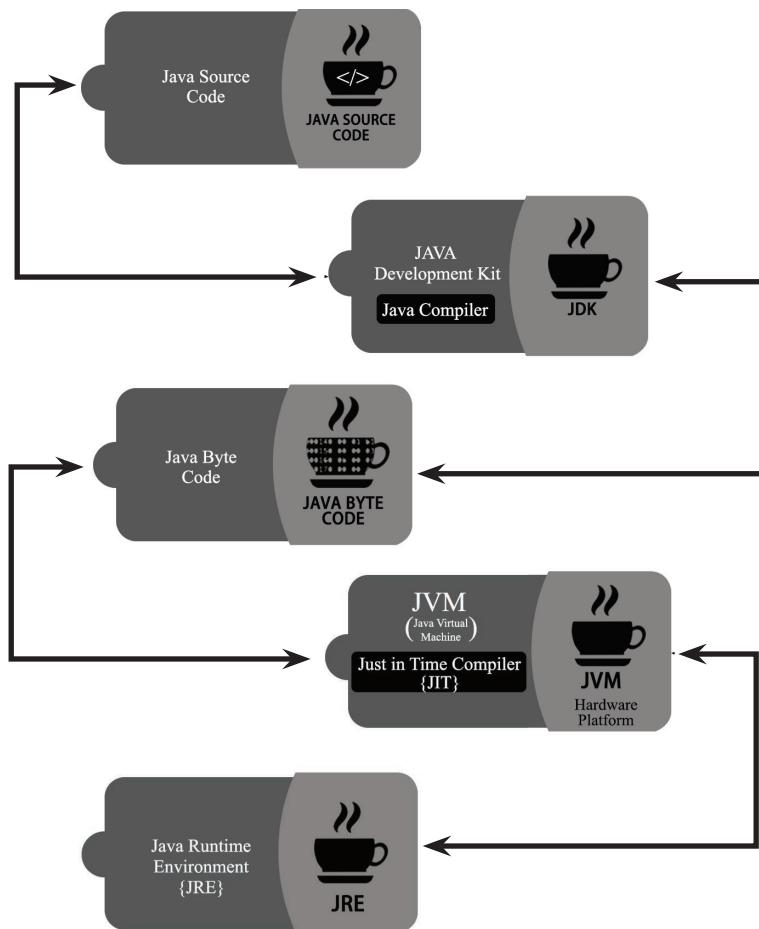


Figure 10.2 Java program execution flow through JDK, JVM and JRE.



10.2.5 Garbage Collector

One important benefit of Java over other languages is the unique way in which it addresses memory allocation and use. In fact, one of the most notable improvements in Java 9 is to specifically improve the memory handling, termed as garbage collection (GC). The JVM creates a memory space for each object from a defined memory heap.

Since a pool of storage is already available, a program in Java can quickly start on any device with a JVM. However, a garbage collector always runs in the background to keep a tab on how the available memory is employed. It checks the use of objects and reclaims the memory from objects which are no longer required by the Java application.

Java provides implicit memory management, which eliminates the need to implement any code that relates to performing the required memory management within the program. Memory management is a fundamental Java feature, which gives the program an edge when using programs that can be scaled on different levels to offer the same level of service and functionality.

Figure 10.3 shows how garbage collector helps in cleaning the orphan object references. We will cover this concept in more detail in Chapter 17.

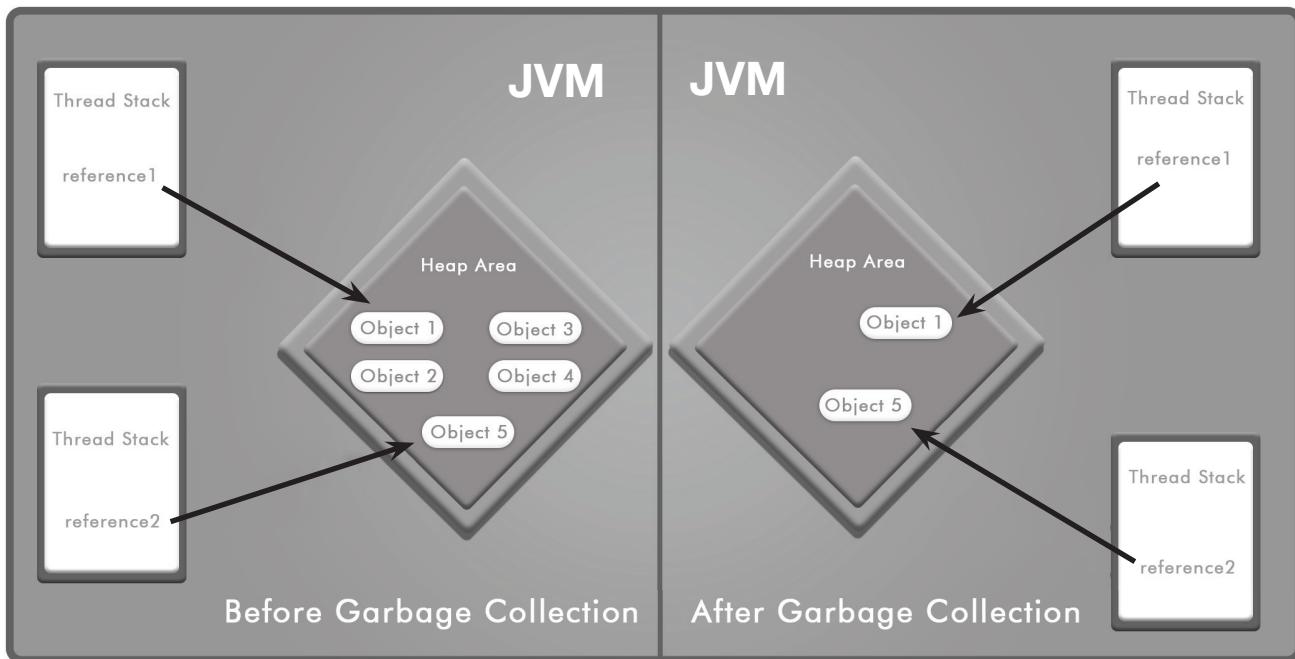


Figure 10.3 Objects state after execution of garbage collection.



If a variable is accessible by a thread, will it get collected by the garbage collector?

10.2.6 Objects in Java

Java is an object-oriented programming language and is different from its parent language C from which it borrows its main syntax and structure. It is a language which combines operational instructions with the required data elements into what we term as objects. These objects are instances that contain the complete structure which describes both the attributes of the functionality as well as how it will behave in all environments.

Objects allow the use of program logic in an integrated manner, and therefore make Java ideal to implement high-level logic and programming potential. The objects are used for all kinds of functions in Java. Objects often consist of subclasses like parent or child objects.

There are various important concepts about objects that we will discuss when describing the program syntax and how to write the Java code.

10.3 | Principles of Object-Oriented Programming in Java

As discussed earlier, Java is an object-oriented programming language (OOP). Every Java program follows these important OOP principles. The OOP principles are designed to combine the attributes of a function, with the required behavior in a single package, which is termed as an object. In the following subsections we will define four important OOP principles that you will observe throughout each course and tutorial available on Java programming – Abstraction, Encapsulation, Inheritance, and Polymorphism.

Before we begin with the principles lets first understand two important concepts IS-A and HAS-A relationship.

10.3.1 IS-A and HAS-A Relationships

IS-A and HAS-A relationships define how different classes are related to each other. Let us explore these concepts in detail below.

10.3.1.1 IS-A Relationship

IS-A relationship exists when one object is type of a particular class. In other words, the given object IS-A class. For example, Cat is a type of Animal so you can say that Cat is an Animal. Cat got all the properties of Animal and a little more properties of its own. Another example would be, Toyota Corolla is a type of Car so it is safe to say that Toyota Corolla is a car. Let us see the following example which shows this relationship.

```
package java11.fundamentals.chapter10;
public class Vehicle { }
```

In the above example, Vehicle is a class which contains the information of Vehicle.

```
package java11.fundamentals.chapter10;
public class Car extends Vehicle{ }
```

The above code shows a Car class extends Vehicle class that means Vehicle is a superclass and Car is a subclass. Hence, as per our IS-A definition, Car is a Vehicle. Now, there can be many types of Cars which will have all the things that Car class has. So let us create a ToyotaCorolla class which extends Car class.

```
package java11.fundamentals.chapter10;
public class ToyotaCorolla extends Car { }
```

Hence, as per our definition, we can safely say that ToyotaCorolla is a Car. Now, let us look into the definition of these classes in terms of Object Oriented terms.

In object-oriented terms, the class relationships can be defined as follows:

1. Vehicle is the superclass of Car.
2. Car is the subclass of Vehicle.
3. Car is the superclass of Toyota Corolla.
4. Toyota Corolla is the subclass of Vehicle.
5. Car inherits from Vehicle.
6. Toyota Corolla inherits from both Vehicle and Car.
7. Toyota Corolla is derived from Car.
8. Car is derived from Vehicle.

9. Toyota Corolla is derived from Vehicle.
10. Toyota Corolla is a subtype of both Vehicle and Car.

According to IS-A relationship, the following statements are applicable:

1. “Car extends Vehicle” means “Car IS-A Vehicle”.
2. “Toyota Corolla extends Car” means “Toyota Corolla IS-A Car”.

10.3.1.2 HAS-A Relationship

As we have seen in Section 10.3.1.1, IS-A relationship is based on inheritance. IS-A relationship is based mainly on how one class is related to the other. HAS-A relationship is different as it is about usage and not inheritance. In other words, how one class is used in another class. Let us see the following example to understand this better.

```
package java11.fundamentals.chapter10;
public class ToyotaCorolla extends Car {
    public static void main(String args[]) {
        NavigationSystem ns = new NavigationSystem();
        ns.getDirectionInstructions("New Delhi", "Agra");
    }
}
```

The above class ToyotaCorolla uses another class named “NavigationSystem” and then calling its method “getDirectionInstructions”. See the following NavigationSystem class.

```
package java11.fundamentals.chapter10;
public class NavigationSystem {
    public String getDirectionInstructions(String from, String to) {
        return "Directions from " + from + " to " + to;
    }
}
```

In the above case, ToyotaCorolla and NavigationSystem are not related by inheritance but by usage. NavigationSystem object is used in ToyotoCorolla object to get direction instructions from one place to another.

10.3.2 Abstraction

Abstraction is the idea of revealing what is needed and hiding data that was not intended for the outside world. A simple example would be a car that is presented as a whole unit and not in the form of components. A user interacts with the car and accesses the required components like driving wheel, gears, etc. and not engine and other core components.

Java uses objects as abstract structures that provide usefulness in a variety of situations. Programmers can create all kinds of objects – individual variables, applicable functions, or data structures – for use in different parts of the same program. It is also possible to create classes of objects. This allows specifying the method of an object with the available syntax in Java.

Abstraction is possible because you can create specific classes of variables and datasets by setting up objects. Take the example of using a class that defines addresses. You can set up parameters such as the name of the person, the zip code, and the postal address within the class objects. In turn, different object instances may describe different addresses for a business, such as an employee and customer addresses.

Another real-life example would be an Automated Teller Machine (ATM) which dispenses money, tells us account balance, allow us to change card pin, etc. but we do not know the inner working of the ATM for performing all these operations for us (Figure 10.4).



Figure 10.4 Example of Abstraction.

10.3.3 Encapsulation

All objects are discrete, which means that all objects carry all the necessary information in their structure to execute successfully. This concept is termed as an encapsulation. The details and specifications of the objects remain hidden on the outside; this gives them improved protection which is not available with other programming languages.

The objects in Java can have relationships with each other, but they always have a boundary layer that isolates them from the external language environment. There are access modifiers available in Java which provide greater control over the objects, turning their status from public to private. A private object is the one whose attributes can only be read from within the object.

Java programming offers excellent encapsulation control as you can always change the width of the boundary of individual objects. This allows Java programming to offer superior solutions that have protected elements, which can be easily used in different elements thus maintaining their internal structure of attributes.

You can always reuse any function while ensuring the highest security. This is important because it saves time, while still ensuring a private setting. Once a specific code is created in the form of an object, it can be used with any database or process, while still ensuring that the actual information available in the database remains private within each object instance.

The availability of encapsulation allows programming to keep the original code secure, while others to use it for the intended functionality. This is excellent when collaborating with others, which is another strong point in Java programming. The Oracle Community allows you to share your problems and codes with other programmers to implement improvements and new library elements. Refer to Figure 10.5.

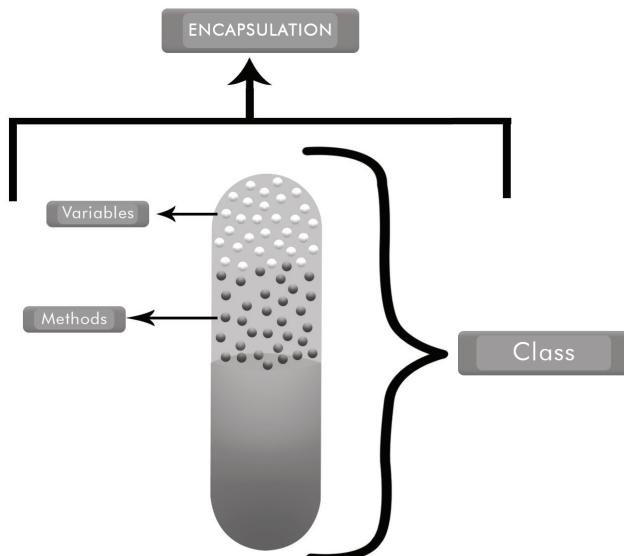


Figure 10.5 Explanation of Encapsulation.

10.3.4 Inheritance

Copying a code segment and changing its attributes creates a new record. By performing this copying and altering process multiple times, a program may have multiple instances of having a duplicate code, which requires greater maintenance and may also affect the performance of the program. This situation is resolved in OOP available in Java by implementing inheritance. This is a concept, where programmers can employ special classes which allow them to copy any attributes and the behavior of the source class. Programmers can simply override the attributes that they need to change. This creates a parent-child structure, which we will further describe when we discuss how objects relate to each other.

The main class here will be the parent one. All copied instances from this special class will be the child objects that use the same structure but have different attributes according to the requirements of the program.

Take the example of a class available for recording information of people. The different child classes may include employees, directors and managers, which may have the People class attributes as well as additional attributes that define their particular information. This principle of OOP eliminates the duplication of the code, while also removes the manual efforts required in copying and then changing all the attributes of an available class.

This creates a parent child structure as shown in Figure 10.6, which we will further describe when we discuss how objects relate to each other.

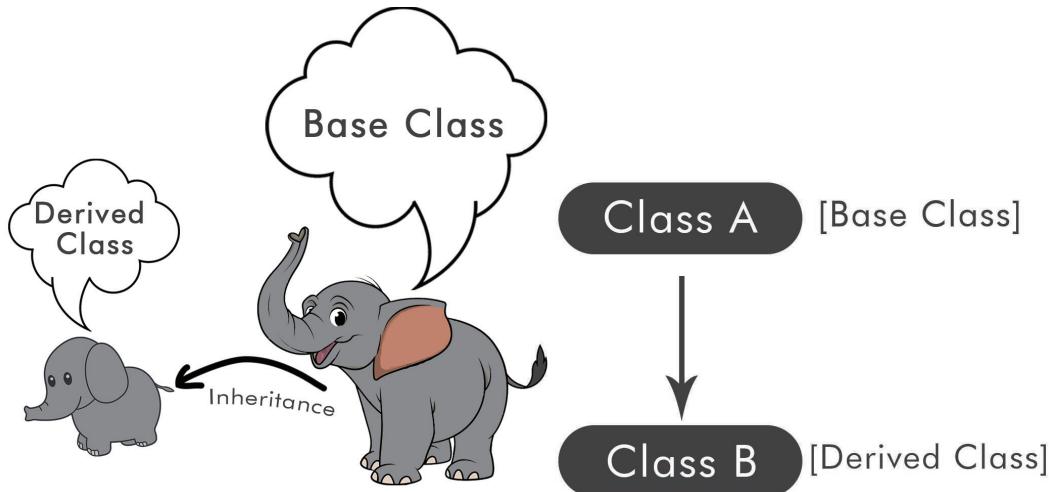


Figure 10.6 Explanation of Inheritance.

10.3.5 Polymorphism

Let us try to understand the meaning of the word polymorphism better. As per the Merriam-Webster dictionary, the word “polymorphism” can be divided into “poly” meaning “many : several : much” and “morphism” meaning “quality or state of having (such) a form”. In OOP terms, it is the object’s ability to take on many forms. Any class object that passes more than one IS-A test is considered polymorphic. With polymorphism, parent class object can be used to refer to the child class object. Let us define polymorphism a little differently to understand it better. Polymorphism is a special OOP concept that allows different objects of the same hierarchy to show separate behavior when they receive a similar message. The term polymorphism refers to being changed into various forms, which is what occurs when objects receive the same message.

It is a complex process often employed at higher programming levels. It allows detailed use of available data in different objects. The way objects respond is subject to how they can be employed in a specific situation.

Consider the following example shown in Figure 10.7 to understand how polymorphism is applicable for a real world situations: Walk is an action which may have different types. For example, you may walk alone, walk with someone, walk with iPod, etc. Hence, the implementation of all these will be different. This shows that one action can have many forms. This is what is known as polymorphism.



Figure 10.7 Example of Polymorphism.

Now, let us take another example shown in Figure 10.8 which is very common. In this example, you can see there is a parent class called Animal and three child classes named Dog, Cat, and Duck. They have a common method named Sound() that they inherit from the parent Animal class. Although they are all Animal family but each class's implementation of Sound() method is different like Dog says Bow Bow, Cat says Meow Meow, and Duck says Quack Quack. And you can refer any child object with Animal object.

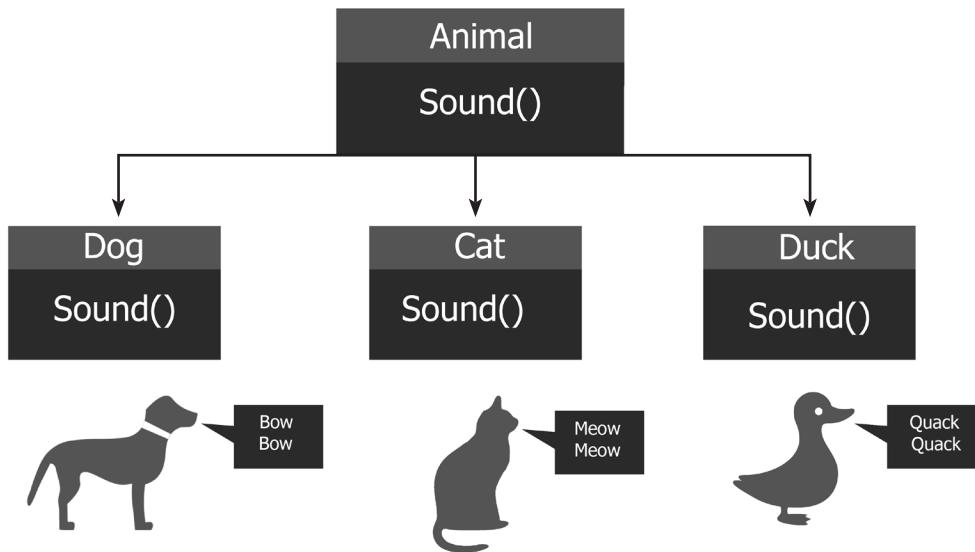


Figure 10.8 Another example of Polymorphism.

10.4 | Programming in Java



You can program in Java once you have the Java Development Kit installed on your computer. Although most text editors are capable of creating Java code files, it is important that you know where the java.exe and the javac.exe application files are present to compile the code that you can create in any editor.

You can also use an IDE for creating Java programs. We will discuss more about IDE it the last section, where we specifically describe the use of Eclipse for creating, compiling, and completing Java-based programming projects. Here, we focus on how you can start to program by first installing a JDK, learning about the language syntax, and the basic programming tools employed in Java.

10.4.1 Installing a JDK

JDK is available from Oracle for all platforms. Whether your computer runs on Windows, Linux, Mac, or Solaris, there is a JDK available free for you from the Oracle website. Once you have downloaded the JDK, you are all set to start programming. There are several programs that you can employ in addition to Java to help you program for specific environments such as preparing web pages, server controls, or database software.

The next step may be to install an IDE with which you are comfortable. We recommend using Eclipse which offers excellent programming advantages, as well as the ability to perform project management. Remember, simple Java programs and applets can be compiled and executed from the command prompt as well. However, the use of an IDE offers numerous advantages and the ease of programming that we will further discuss with Eclipse.

10.4.2 Using Libraries

The JAR files can be easily employed to enjoy the functionality offered by different Java class libraries. Your libraries are easy to add in your program. Simply add all the libraries that you intend to use on the path which describes the available Java project when using an IDE such as Eclipse. For new programmers, Java libraries contain the codes that you need to ensure that your Java program functions. They are implemented using JAR files.

A JAR library file is an archive of zipped files. Each JAR contains a collection of Java classes and other relevant sources, such as property files and icons. These files are designed to be executable when you define them in your classpath in an IDE or mention them in your programming code.

Remember, you must have the required JAR file referenced in your program if you are using different classes that are available for use in Java. Another concept is the use of executable JAR files, which allow the user to run an application without having to individually mark the Java classes which need to be initiated for a program.

In fact, the JVM must always be informed about the path of the *main()* method which is always employed in a Java program. You can easily create your own JAR files using a command:

```
jar -cvmf MANIFEST.MF myawesomeapp.jar *.class
```

This is possible by simply using a command prompt. Since we are already describing the use of Eclipse in Section 10.7, we will not focus too much attention on describing the use and the implementation of library-based classes as implementation is easy to carry out in the IDE.

10.4.3 Syntax

A key element to learn when programming with Java is the syntax of the language. The open-source programming language takes its basic syntax from C and simplifies it with the ability to objectify the code into simple blocks which are consistent and can be reused throughout the program. Remember, the language syntax is a set of rules that define how the compiler will read the information written in a text editor or an IDE.

Although Java syntax is based on C language, it does not employ global functions. It consists of data members like class level variables that always act as global variables for the class and are accessible by the class's objects in other classes.

Java employs separators of "{}" to describe a specific code block. A semicolon ";" indicates the end of a particular code element, often termed as a statement in other languages. A Java code is defined in terms of classes, which act as templates for the functionality that is required from a Java program or its different sections.

Java is a case-sensitive language, where the words, say, program and Program mean differently to the compiler and can represent separate identities. Another practice in Java syntax is that all class names must start with a capital letter. If you are using multiple words for a class, then every inner word should start with an upper-case letter. A right example of this will be to name a file as "MyFirstProgram", while a wrong practice would be to name it as "myfirstprogram".

The next syntax that you must follow is that every method, often termed as a function in other programming languages, must start with a lower-case letter. However, if it is a large name with several words, each successive word should start with an upper-case letter. An ideal example is "myMethod", while a poor one is "Mymethod" for naming methods.

Remember, your file name should be exactly the same as the class that you created. This means that according to our earlier example, you will name your file as "MyFirstProgram.java". Also remember that you need to append .java at the end of your file name to create files that show that they contain Java code and can be read by the Java compiler to create a JVM compatible Java bytecode file.

The processing always starts from the `main()` method, which is a programming syntax derived from the C language structure. A common way of using this method is in the following form:

```
public static void main(String args[])
public class HelloWorldExample {

    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}
```

The main program code is placed inside the braces that are present in the `main` method. All other Java components are named in a unique manner, termed as the identifiers that we will cover shortly in the subsequent programs.

Now, let us take a little detailed look at the class structure. All Java programs start by declaring a class and then defining different methods within it as required for producing the intended functionality. A common way of defining a typical class would follow this construct:

```
classAccessModifier class ClassName {
    variableAccessModifier variableDataType variableName = defaultValue;
    //Constructor declaration
    methodAccessModifier ClassName(variableDataType parameter1, variableDataType
parameter2,.....) {
        //code
    }
    methodAccessModifier methodReturnType methodName(variableDataType parameter1,
variableDataType parameter2,.....) {
        //code
    }
}
```

Do not be scared of the above program as it is only to show you how Java program is structured. You do not use those keywords in writing a Java program. Let us see a quick example to understand how to use the above structure to write a program.

```
public class MyFirstClass {
    private int myFirstVariable;

    //Constructor Declaration
    public MyFirstClass() {

    }

    //Method declaration
    public int myFirstMethod(int myFirstVariable) {
        System.out.println("This is my first method which prints value of myFirstVariable :"
+ myFirstVariable);
        return myFirstVariable;
    }
}
```

This defines almost all Java programs regardless of their size, complexity, and the intended functionality. A common specification will always be to use the public access, as this allows all code in the Java program as well as an import to access the class, the variable, or the methods employed within the program.

10.4.4 Declaration Rules

You have already seen how a sample program looks like. Now is the time to see rules that are applicable to source code file creation.

1. Only one public class per source code file is permitted
2. Comments can appear anywhere, beginning or end.
3. Name of the source code file must match the name of the public class and end with “.java”. For example, in our HelloWorldExample, we have HelloWorldExample as our public class and hence the file name must be HelloWorldExample.java.
4. Although it is not mandatory to create separate packages, it is a good practice to create separate ones. In that case, package declaration must be the first line even before import statements.
5. For any imports, import keyword must be used followed by the class full path. And these statements must appear between package declaration and class declaration.
6. There can be multiple class declarations but only one class can be classified as public. Hence, these import and package statements are applicable to all the packages and imports.
7. A file which contains the program with no public class can have any name that does not need to match any of the classes in the file. Only public class name must match with the file name and a file can have only one public class.

QUICK CHALLENGE

Create a source code file, give it a name, and use the same name for the class. Mark this class as public. Now, add another class in the file and mark that as public as well. What will happen in this case?

10.4.5 Identifiers

Identifiers are the proper nouns in Java programming language, which can be used to denote any language structure. There are unique names for methods, classes, packages, interfaces, and variables in Java. In fact, every word other than the literals and language keywords can be termed as identifiers. Take our example where all names like `main`, `args`, `String`, `println`, and `MyVeryFirstJavaProgram` represent identifiers. However, they all represent different Java elements which we will describe in their relevant sections.

There are some rules to be followed when setting up unique identifiers in Java:

1. The first rule is that we cannot use a space within a name in Java. Spaces between words cannot be identified by the compiler and therefore, remain restricted.
2. Another rule is that we cannot start an identifier with a number. However, it is possible to start with an underscore, although it is not the ideal practice. There are some other characters which should not be employed such as hyphen, apostrophe, and ampersand.

There are also some set naming practices which provide a convention. Class names start with a capital letter, while methods always start with a small letter. This policy ensures that it is easy to find out a mistake in the code, while also ensuring that any programmer looking at your code can quickly identify the type of a unique element, without looking around for the neighboring code elements and statements.

Another point within this is to remember that Java identifiers are always case-sensitive, and it is not possible to keep using different cases, as the compiler will not understand a case use, and will generate an error. The last point is that Java has several keywords which it uses to represent specific meanings and literal values. You cannot use these keywords as identifiers in your Java programs.

QUICK CHALLENGE

Give some examples of identifiers.

You must follow these rules when naming your identifiers:

1. All identifiers must begin with \$ (currency), _ (underscore), or any alphabet (lower- or upper-case). Examples of valid identifiers are `$myVariable`, `_myVariable`, `myVariable`, `myVariable$`, `my_Variable`.
2. After the first character any character is allowed to name an identifier.

3. A Java keyword cannot be used as an identifier.
4. Java keywords are case-sensitive.

Java also uses modifiers. As you saw earlier, the word “public” was used to define the main method. There are two types of modifiers: (a) *access modifiers* that include public, private, protected, and default values and (b) *non-access modifiers* that include abstract, final, strictfb values.

Let us explore these modifiers next.



10.4.6 Access Modifiers

Now, let us see how access modifiers look from a microscopic point of view. When we talk about access modifiers, we talk about how much we need to restrict or allow access to a class, method, or variable.

Java offers four types of access modifiers but only three have names – public, private, and protected. The fourth one does not have any name and it is known as default or package level access. Default or Package level access means every class, method, and instance variable has access control, whether we assign or not. Now here it gets interesting: class can be of public or default access as other two modifiers do not make sense. You may declare an inner class as private.

Now, let us have a look from the class point of view. Let us see how these modifiers are useful for classes.

10.4.6.1 Class Level

1. **Default access or package level access:** As we have just seen, the default modifier does not have any modifier name before it. This modifier is known as package level modifier as the class with this type of modifier is only seen from the same package classes. For example, if the class myClass is in the package called com.fullstackdevelopment.mypackage and has no modifier in front it then this class cannot be accessed by the class yourClass which resides in other package named com.fullstackdevelopment.yourpackage. Since these two packages are different, yourClass has no access to myClass.
2. **Public access:** A class that is declared as public is accessible by all the classes from all the packages. This is the most used access modifier for a class. In the above scenario, if we change the access modifier for myClass as public, then myClass becomes available to yourClass even though it is residing in two different packages.

10.4.6.2 Method and Instance Variables Level

Method and instance variables can use access modifiers in the same way. As we have seen earlier, class can only use two of the access modifiers; however, methods and instance variables can use all four – Public, Protected, Default, and Private. See Figure 10.9 which shows the access modifier levels.

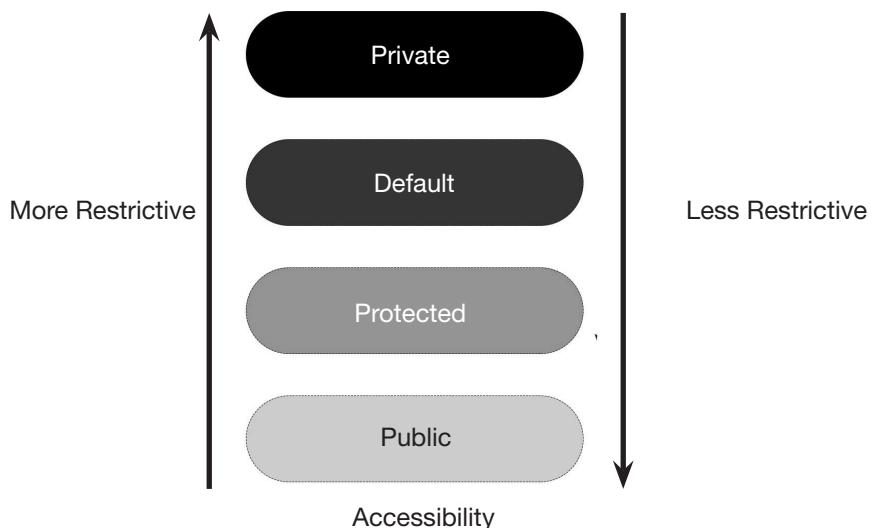


Figure 10.9 Access modifier levels.

Following modifiers are applicable to methods and variables. Public and default modifiers are similar to class modifiers that we have just seen above.

1. **Default access modifier:** When we do not declare any modifier, we get default protection which is at package level.
2. **Public access modifier:** Public access control is much wider because any class that has access to the class that contains public methods and instance variables can access these methods and instance variables.
3. **Protected access modifier:** Protected access modifier works quite similar to default modifier except for one difference that is default modifier methods and instance variables are accessible to the package only whereas protected methods and instance variables can be accessed via subclass even though subclass is in a different package.
4. **Private access modifier:** Private is the most restrictive access modifier. This modifier is only accessible by the same class members. No other class, even a subclass, can access private members of the superclass.

Table 10.2 summarizes the visibility for the access modifiers.

Table 10.2 Access Modifiers Comparison

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, <i>through inheritance</i>	No	No
From any non-subclass class outside the package	Yes	No	No	No

To understand this better, let us see the same table in a different way:

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

It is important to understand the concept of modifiers and how their scope is defined. Figure 10.10 shows the modifiers scopes and how they are limiting the access for identifiers.

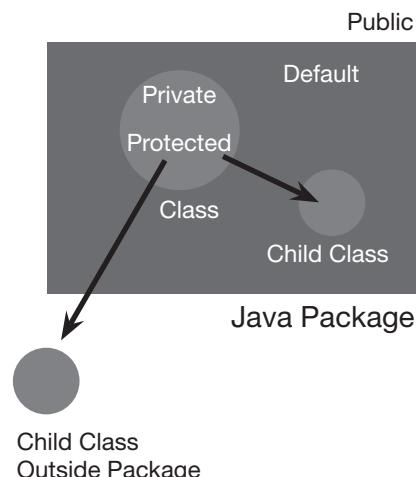


Figure 10.10 Access modifiers scopes.



10.4.7 Non-Access Modifiers

Class can also be declared with non-access modifiers such as final, abstract, or strictfp. These non-access modifiers are additional modifiers you could use with access modifiers. For example, a class can be public and final. You could also use two non-access modifiers such as final and strictfp. However, it is not at all advisable to use final and abstract. Let us see final and abstract modifiers in detail below.

- Final modifier:** This modifier makes the marked class non-inheritable. In other words, the class marked as final cannot be subclassed that is it cannot be extended by any other class. If tried, it will throw a compiler error. Now this might make you think, why would we even think of marking a class as final if we cannot extend it? Isn't this violating the object-oriented principal? Yes, you are right but there are circumstances when you want an absolute guarantee that the class's behavior stays intact and no one can make changes to any method. In this scenario, final comes handy. Some of the core Java libraries are final, for example, String class. Final gives the guarantee to the String class that the class functioning is intact. Example

```
public final class MyFirstClass {  
}
```

Now, no class can subclass MyFirstClass.

- Abstract modifier:** A class marked as an abstract can never be instantiated. The whole purpose of having an abstract class is to be subclassed. Now, why would you have a class that cannot be instantiated? One reason could be that the class is too abstract, and you want the subclasses to extend it properly. For example, a car could have some generic methods that a typical vehicle can have. But we may want very specific cars with very specific features and not generic like this abstract car. This is when abstract modifier is useful.

If you declare a method as abstract, you must declare a class as abstract as well. However, an abstract class may have any or all non-abstract methods. Figure 10.11 shows the rules of using Abstract class.

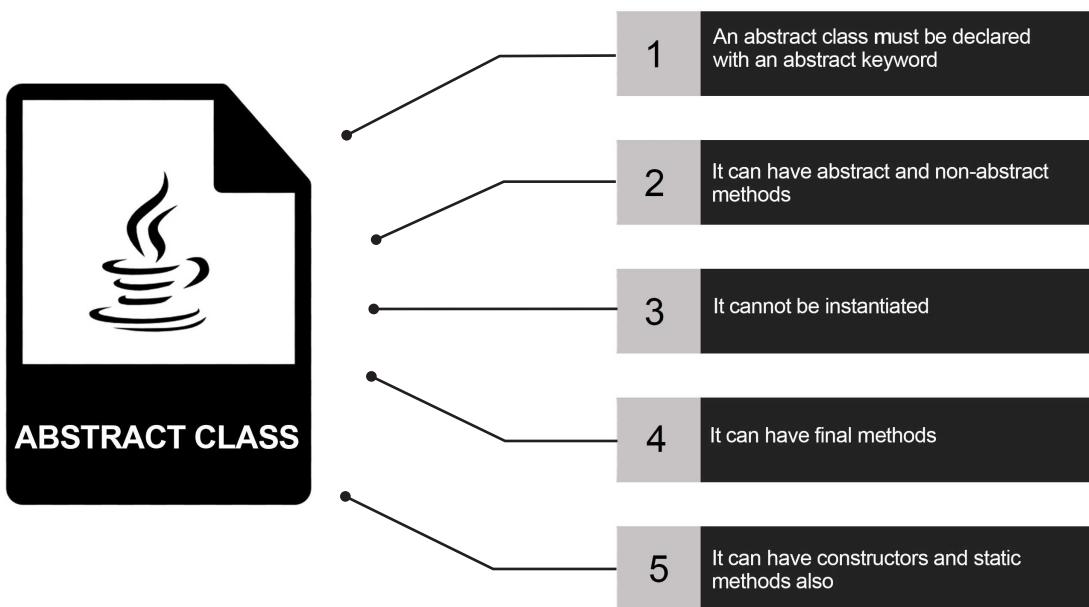


Figure 10.11 Rules for Java Abstract class.

If you declare a method as abstract, you must declare a class as abstract as well. However, an abstract class may have any or all non-abstract methods.

10.4.8 Reserved Words

As with every high-level programming language, there are several words that are reserved to represent important language commands and built-in functions. Here is a list of the words that you are not allowed to use when naming different elements in your Java program. They are reserved for Java use, and you will often employ them in programming in the same capacity:

1. abstract	18. final	35. public
2. assert	19. finally	36. return
3. Boolean	20. float	37. short
4. break	21. for	38. static
5. byte	22. goto	39. strictfp
6. case	23. if	40. super
7. catch	24. implements	41. switch
8. char	25. import	42. synchronized
9. class	26. instanceof	43. this
10. const	27. int	44. throw
11. continue	28. interface	45. throws
12. default	29. long	46. transient
13. do	30. native	47. try
14. double	31. new	48. void
15. else	32. package	49. volatile
16. enum	33. private	50. While
17. extends	34. protected	

Java also has three constructs of null, true, and false which you cannot use because they are read as literals by the language compiler. An excellent advantage of programming in an IDE is that it uses a color scheme that quickly allows you to identify that you have written a Java reserved word in your code. See the following example which shows the reserved words in purple color that includes package, class, private, int, public, and return.

```

1 package com.fullstackdevelopmentwithmayurramgir;
2
3 public class MyFirstClass {
4     private int myFirstVariable;
5
6     //Constructor Declaration
7     public MyFirstClass() {
8
9     }
10
11    //Method declaration
12    public int myFirstMethod(int myFirstVariable) {
13        System.out.println("This is my first method which prints value of myFirstVariable :" + myFirstVariable);
14        return myFirstVariable;
15    }
16
17 }
```

10.4.9 The Keyword “this”

The keyword “this” refers to the current object. Following list shows the use of **this** in any Java program.

1. Current class instance variable can be referred by **this**.
2. Current class method can be invoked implicitly by **this**.
3. Current class constructor can be invoked by **this**.

4. A method argument list can contain **this**.
5. A constructor argument list can contain **this**.
6. A method can return **this** which refers to the current class instance.

10.4.10 Classes and Objects



Java classes and objects form the basis of the OOP concept in the language. The first principle that defines a Java class is that it should have singular functionality. This allows the use of class or particular objects whenever a particular need arises without having to write new code. This principle is called as “Cohesion”. Cohesion is the degree to which a one particular class has a single, exclusive purpose. If a class has high cohesion, it means the class is focused on a single functionality and only useful to serve a particular purpose.

To understand this better, we give an example of a Student class and its objects. Student is a class which only focuses on student-related functions and it has student-related states and behaviors. All the objects that are deriving from this class get these student-related states and behaviors and do not carry any other functionality.

Figure 10.12 shows a Student class that has states like Name, Age, Color, and Sex. Student class also has behaviors like Eating, Drinking, and Running. Every object derived from this particular Student class gets these states and behaviors. The left side of Figure 10.12 shows Objects which have their states defined like first object has Name as Roshni, Age as 11, Color as Dark and Sex as Female. Similarly, other objects get the states and can have their own values for those states.

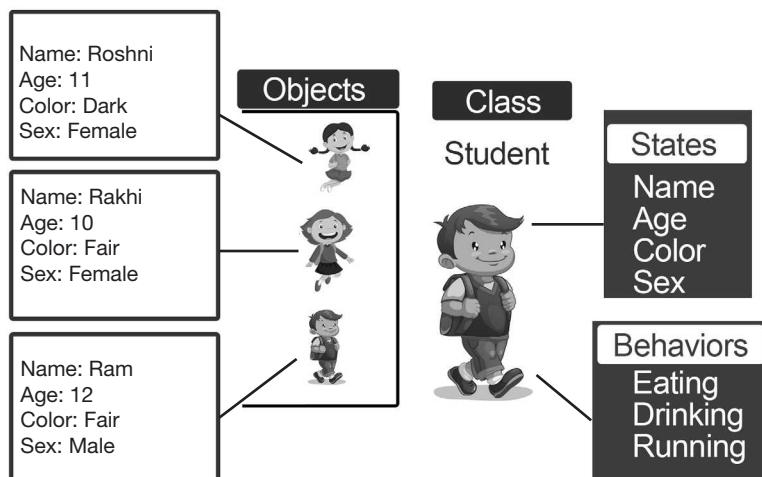


Figure 10.12 Objects and Class example with States and Behaviors.

Another property that a Java class has is an open, closed design. This is a design which describes that the classes and the methods present in them must remain closed to prevent modification. However, they should be open to an extension where the current code present in the class can be used to implement further functionality in a Java program.

A Java class works as a blueprint for another important concept, which is the use of objects that function as discrete blocks of code in the programming. Your Java application uses object instances. It is designed to carry out the functionality required of it, in different sections of the program. Java packages control the way naming conflicts are resolved, and therefore a namespace is always uniquely created when compiling a program in Java.

Java packages are named using a standardized scheme. This improves control over generated classes and allows programmers to use the available packages and libraries. In fact, if you name all your classes in a unique manner, you will never have to mention a package to describe them.

The working part in Java programming is an object. It is often a particular class case which has its own definition and behavior according to the inputs it receives. The Java objects often follow a simple hierachal structure, where a parent object serves as the basis for other objects (child) that come under it. This structural exercise allows you to create a set of objects that have the same definition but can show different behaviors. A key feature of Java objects is their ability to communicate with each other. This ensures that a program runs in a coordinated manner and can perform tasks in a systematic manner, especially in relation to the specific case that may appear during each program execution.

Java objects must perform a definite set of activities. They should only declare or hold their own data in an ideal scenario. However, they may need access to other objects if they are employed in carrying out the defined activities within their definition. An object must only have a minimalistic set of dependencies that are sufficient for it to perform its activities.

Take the example of an object that defines a person. Such an object in line with the Java object definition will have a set of two elements – the attributes and the object behavior. The attributes are often termed as the nouns that define an object. A Person object may have attributes such as age, name, contact details, gender, and other elements that may present the definition of the described object.

The behavior of an object may be best understood as the action verbs that it can perform. Let us follow our example of a Person object. A behavior would be to use the available attributes to calculate the BMI. Another verb in this manner would be to print all attributes for a particular object instance. The state of the current attributes can also be found as a behavior, where the object can return the current values using a String.

10.5 | Java Packages

Most Java programs start by mentioning a Java package. Java packages represent collection of classes and their related code elements. Packages also provide the option of using access modifiers. A package is present at the start of the program file and provides a dedicated path for the storage of the program class.

10.5.1 Structure

Defining a class is important. You use it to define various objects, and in turn, Java application will run object instances on execution. There is an import statement present in a class definition as well. This allows the compiler to find the classes that you use in your code. One class may possibly use other classes, and it necessitates that the compiler is aware of where to read them for execution. The structure of the import is as follows:

```
import ClassNeedsImport;
```

The class name should always include its package as well because it is required for an independent qualification. There are times where you may need multiple classes that you are using in your non-trivial class. It would not be an ideal practice to name every individual class separately. You can import a set of classes as well. By using a “*” after a package name in the import statement, you can import every available class from the package for use in the program. Here is an example:

```
import com.javapackage.*;
```

A good practice is to only import the classes that you need by using their qualified names. This keeps your code easy to read and ensures that you are following the minimalistic structure, which is the basis of using an OOP language like Java. For any object that you need to create, you must first declare its class. This works out like creating a template which can then be employed multiple times to provide the same functionality.

A common class declaration is defined in the following manner:

```
classAccessModifier class ClassName {
    variableAccessModifier variableDataType variableName = defaultValue;
    //Constructor declaration
    methodAccessModifier ClassName(variableDataType parameter1, variableDataType
parameter2,.....) {
        //code
    }
    methodAccessModifier methodReturnType methodName(variableDataType parameter1,
variableDataType parameter2,.....) {
        //code
    }
}
```

The specifier remains public. The class starts with an upper-case letter. This allows all programmers who may look at your code to easily understand the required functionality of the class, as well as the objects that are created through it. A class heavily depends on the variables that allow each instance of the class to be distinguishable and fully separate from one another.

Figure 10.13 shows how classes are defined. As you can see in the left box, a class definition contains variable, methods, and statements inside methods. And in the right box, you can see the code syntax for defining a class. In this example, 'type' can be any primitive, non-primitive or object type.

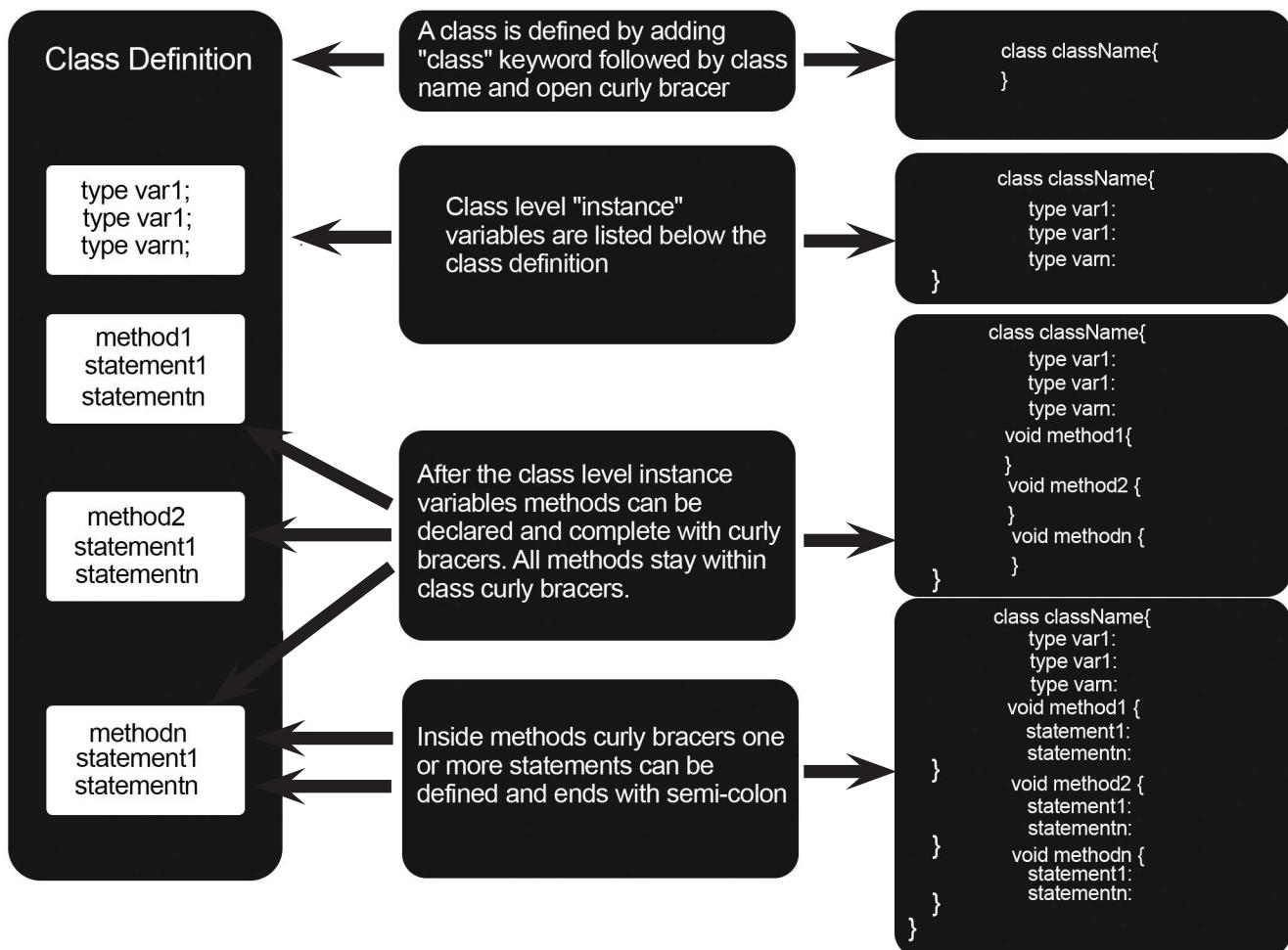


Figure 10.13 Class structure and elements.

10.5.2 Variables

Each class is defined by a set of class variables. They declare the specific state of the class in its different instances. This enables the use of the class elements in a variety of settings. The first element in a variable is an access modifier. This can have any state – from the group of public, private, protected, and no specifier. If no specifier is used, the variable is available for the entire Java package.

The next part is the data type, which defines whether it is a primitive class variable or belongs to another type. The variable name is the next part, which should follow a camel case. This is a convention which starts with a lower-case letter. Programmers can also define an initial value for the variable. This is just like the initialization that you can use with variables that are created in other programming languages. The compiler sets up an initiating value for the class variables if they are not predefined. A class would often have a set of variables in the following manner:

```

public class GymCustomer {
    //In the following line 'private' is an access modifier, 'int' is a primitive data
    type and 'age' is a variable
    private int age;
    private int height;
    private String name;
    private String gender;
}

```

The above code contains a class with the name GymCustomer that has a set of attributes in the form of variables. There are two data types mentioned in these variables. The first are the integers which describe the use of whole numbers, while the second one is String designed to record character strings such as the name, gender, and skin tone of the GymCustomer class.

10.5.3 Methods

The information about a particular class is incomplete without the use of methods. Methods are behaviors that the class objects can show. There are two categories of methods. The first category is that of the constructors. All other methods such as void and the ones which return value form the second category and are used to signify any customized behavior which is required of a particular class instance.

As visible with our earlier example, methods also need to have their access specified. They also require a type of return, which signifies the value that they will produce. The name is mentioned next, while the parentheses contain the arguments required for the intended functionality. The complete text that contains a collection of these details is termed as the signature of the method. The infographic shown in Figure 10.14 can help us understand this concept a little better.

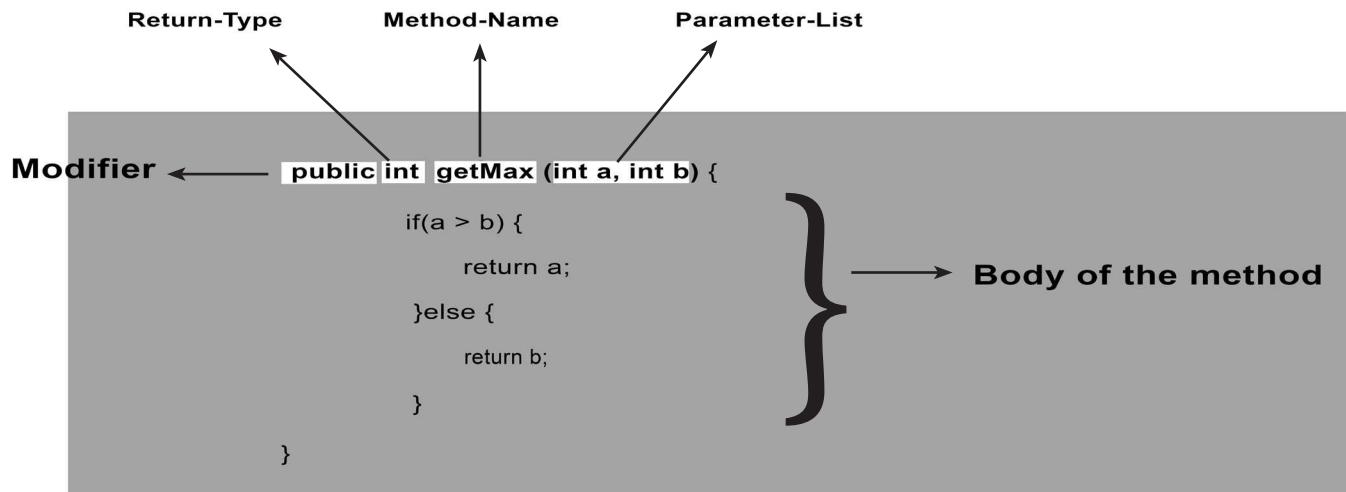


Figure 10.14 Methods.

10.5.4 Constructors

Constructors are methods that describe the class in the form of different statements. The compiler automatically uses a no argument constructor for your instantiating needs. This makes them optional. However, if you use a customized constructor, then the compiler will not create one for the specific purpose.

A constructor is always named after its class. It looks like a method but without a return type. Default constructor is the one which does not take any parameters and non-default constructor is the one which takes parameters in other words argument list. Constructor can have similar type of access modifiers as methods can have. If no modifier is given then it is given a package level access. If the constructor in a class is defined as private, no class can instantiate this class as it only gives class level access. The only way to create objects of this class is by having a factory method in the class that calls the private constructor. You can also use this constructor to create the initial attributes. Let us move forward with our earlier example:

```
package java11.fundamentals.chapter10;
public class GymCustomer {
    private int age;
    private int height;
    private String name;
    private String gender;
    public GymCustomer() {
        // Nothing happens here
    }
}
```

In the above example, we have seen “no argument” constructor which is also called as default constructor. The following example shows a different type of constructor where we accept the default values for the member variables while object creation.

```
package java11.fundamentals.chapter10;
public class GymCustomer {
    private int age;
    private int height;
    private String name;
    private String gender;
    public GymCustomer() {
        // Nothing happens here
    }
    public GymCustomer (int age, int height, String name, String gender) {
        this.age = age;
        this.height = height;
        this.name = name;
        this.gender = gender;
    }
    public void printGymCustomerData(){
        System.out.println("Name : " + this.name + " - Gender : " + this.gender + " - Age
: " + this.age + " - Height : " + this.height);
    }
}
```



Can you define constructor as private?

This constructor uses “this” keyword which signifies this object. It allows the use of variables that have the same name. The age, for example, is a class variable, but it is also used as a parameter in the class constructor. The use of “this” allows the compiler to differentiate between these elements with the same name. Following is the example of using these constructors.

```
package java11.fundamentals.chapter10;
public class GymCustomerTest {
    public static void main(String args[]){
        //this will call the no argument constructor to create GymCustomer object
        GymCustomer gc = new GymCustomer();
        // this will call the argument constructor with default values set for the member
variables
        GymCustomer gcc = new GymCustomer(45,5,"Mark Smith","Male");
        gcc.printGymCustomerData();
    }
}
```

The above program will print the Gym customer data like name, height, age, and gender. See the following output.

Name : Mark Smith - Gender : Male - Age : 45 - Height : 5

10.5.4.1 “this” keyword

The “this” keyword is useful to call other constructors in the same class. This type of invocation calls an explicit constructor invocation. See the following example which shows three constructors and first two constructors call the third one explicitly.

```
package java11.fundamentals.chapter10;
public class ThisExample {
    private int x, y;
    private int a, b;
    public ThisExample() {
        this(0, 0, 1, 1);
    }
    public ThisExample(int a, int b) {
        this(0, 0, a, b);
    }
    public ThisExample(int x, int y, int a, int b) {
        this.x = x;
        this.y = y;
        this.a = a;
        this.b = b;
        System.out.println("Printing from the Third Constructor - x : " + x + " - y : " +
y + " - a : " + a + " - b : " + b);
    }
    public static void main(String args[]) {
        ThisExample te = new ThisExample();
        ThisExample te2 = new ThisExample(2,2);
        ThisExample te3 = new ThisExample(3,3,3,3);
    }
}
```

The above code produces the following result. This shows how the call passes through the constructor using “this” keyword.

Printing from the Third Constructor - x : 0 - y : 0 - a : 1 - b : 1
Printing from the Third Constructor - x : 0 - y : 0 - a : 2 - b : 2
Printing from the Third Constructor - x : 3 - y : 3 - a : 3 - b : 3



What will happen if you do not define any constructor?

10.5.4.2 “super” keyword

“super” keyword refers to the super class. “super” is used to invoke the superclass’s constructor. It is only allowed to use on the first line or else compiler will complain. See the following example which shows the use of “super” keyword.



```

package java11.fundamentals.chapter10;
public class SuperExample {
    SuperExample() {
        System.out.println("This is from the SuperExample Constructor");
    }
    public static void main(String args[]) {
        System.out.println("Inside the main method");
        Child c = new Child();

    }
}
class Child extends SuperExample {
    Child() {
        super();
        System.out.println("Child Constructor");
    }
}

```

The above program produces the following output;

Inside the main method
This is from the SuperExample Constructor
Child Constructor

In this example, you can see that from the Child constructor we have made a call to `super()` which calls the super class constructor from SuperExample.

Table 10.3 shows how constructors are different from methods in various areas.

Table 10.3 Difference between constructors and methods

Constructors	Methods
Use	For creating instances of a class
Applicable Modifiers	Constructors can be Private, Public, Default, and Protected but cannot use abstract, final, native, static, or synchronized
Name	It must be exactly same as the class name.
This	this refers to another constructor in the same class. It must be used on the first line in the constructor declaration.
super	super refers to the parent class constructor. It must be used on the first line in the constructor declaration.
Inheritance	It is not applicable to constructors as they cannot be inherited.
Return type	None
	Group code to Methods can use private, public, default, protected abstract, final, native, static, or synchronized
	Any name other than the class name can be used. this refers to an instance of the current class. this cannot be used in static method.
	super is useful to call an overridden method in the parent class.
	Methods can be inherited
	Native data types like int, float, double etc., Native objects like String, Map, List etc., or any other built-in and user defined objects.

10.5.5 Other Methods

The GymCustomer object is ideal to understand the basic syntax of the language. However, you need more behavior implementation. This requires use of more methods that truly create the functionality that a class object needs. Other methods are different because they can have any name (other than the class).

Some important notations, though optional, are great for reviewing your code. You should always start the name with a lower-case letter and avoid using numbers as much as possible. Other methods also allow you to mention a return type, which produces specific functionality for the object. If we continue with our earlier example, we can show how it is possible to use methods other than the constructors. Here, we have omitted the class details and are showing the use of a method after the initial class variables:

```
public String getGender() { return gender; }
public void setGender (String value) { gender = value; }
// May use any combinations like this for information...
}
```

This is the use of an instance where you use two particular methods. The first is the getter method that fetches the value of a variable (attribute), while the setter then allows you to modify that value. You can do this for any attribute which is defined in the class definition. Remember, it is important to mention the type of return in this use. Since the getter method brings a string, its datatype is defined for it. On the other hand, the setter does not return anything, which is then clearly stated using the reserved word “void”.

You can generally use static and instance methods. The static methods are often termed as class methods because they change the behavior at a class level, which is independent of the individual class objects. The instance methods, as their name suggests, are designed to work according to a particular object instance.

Static methods are excellent because they provide utility and ability to offer global functionality, just like the C language. The code used in the static method remains confined within the class that defines this static method. Take the example of a Logger class which is often used to output the information. The static method of `getLogger()` is employed whenever you need the functionality of the particular class.

A static method uses a different syntax since it directly applies to a class rather than individual objects. It is possible to use the name of the class within its use as follows:

```
Logger 1 = Logger.getLogger("NewLogging");
```

The first Logger is the class name and the `getLogger ("NewLogging")` is the static method used. This means that there is no need to create or mention an object instance. The class name is sufficient for creating this type of method use.



Which return type can you use to not return anything from a method?

10.5.6 Programming Examples

In this section, we will discuss some simple examples that will show the basic Java elements that we have described in this particular section. These are simple Java programs that will help you understand the basic functionality that this object-oriented language offers to programmers. Let us start with the typical Hello World program:

```
public class HelloWorld
{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

This is a simple program with the Java class of Hello World. It uses a single method of *main()* which describes the use of a string output. There is only one functionality present in the method, which is to print the message of “Hello World”.

Hello World

Now from this simple example, let us move to something more complex, which uses objects:

```
public class LearnJavaProgramming {
    public void printFeedback() {
        System.out.println("I love Java");
    }
}
public class MyFirstProgram {
    public static void main(String[] args) {
        // creates ljp1 object of LearnJavaProgramming class
        LearnJavaProgramming ljp1 = new LearnJavaProgramming();
        // creates ljp2 object of LearnJavaProgramming class
        LearnJavaProgramming ljp2 = new LearnJavaProgramming();
    }
}
```

This is a simple example where a class *MyFirstProgram* is initiated and then two objects of the same class are added to the *main()* method.

The next example describes the use of methods with objects which bring the real use of the Java structure and its programming methods. The program below from the same source of the above program is an extension that shows the use of methods and calling them to control the class objects in Java:

```
public class LearnJavaProgramming {
    private String feedback;
    public void printFeedback() {
        System.out.println(feedback);
    }
    public void setFeedback(String feedback) {
        this.feedback = feedback;
    }
}
public class MyFirstProgram {
    public static void main(String[] args) {
        // creates ljp1 object of LearnJavaProgramming class
        LearnJavaProgramming ljp1 = new LearnJavaProgramming();
        // creates ljp2 object of LearnJavaProgramming class
        LearnJavaProgramming ljp2 = new LearnJavaProgramming();
        ljp1.setFeedback("First object feedback - I love Java");
        ljp2.setFeedback("Second object feedback - I love Java");
        ljp1.printFeedback();
        ljp2.printFeedback();
    }
}
```

Now you can print the feedback of the two objects. Since the first object *ljp1* is setting its feedback to “First object feedback – I love Java”, it will print the same when *ljp1.printFeedback()* gets called. The same is true for the second object where *ljp2.printFeedback()* will print “Second object feedback – I love Java”.

First object feedback - I love Java
Second object feedback - I love Java



10.6 | New Features in Java 9

There are several new features in Java 9. You can start with the basic that we have covered till now and learn about the advantages of the new version with the information available in this section. Even if you are a beginner, learning about these new features will allow you to focus on the excellent elements in the new version, and include them in your normal programming routines. In the following subsections we will give brief details of the new additions in the latest Java version.

10.6.1 Java Shell

Java was traditionally a language that required programs to be compiled before execution. With the arrival of the Read-Eval-Print-Loop (REPL) functionality through the JShell tool in Java 9, it is now possible to run simple Java code at a jshell console, and instantly execute the output.

Please see the following screenshot of a JShell console, which shows how easy it is to create variables, assign values to the variables, create methods, use the assigned variables in the newly created methods, etc.

```
Mayurs-MacBook-Pro:~ Mayur$ jshell
| Welcome to JShell -- Version 11.0.1
| For an introduction type: /help intro

jshell> String myName = "Mayur Ramgir"
myName ==> "Mayur Ramgir"

jshell> String HelloMethod(String s) {
|     ...> return "Hello " + s;
|     ...>
|     created method HelloMethod(String)

jshell> HelloMethod(myName)
$3 ==> "Hello Mayur Ramgir"

jshell>
```

In the above program, you can see, we started jshell by typing jshell command on the command prompt. With this we get into the jshell prompt. On that, we have declared variable as “String myName” and assign value as “Mayur Ramgir”. It’s the same way we type inside a method as local variable or in a class as class level variable. Upon pressing enter key on the keyboard, jshell registers the variable which we can use later. After that we have created a method “String HelloMethod(String s)” and added a return string “return ‘Hello’ + s”. Again, upon pressing enter key on your keyboard, jshell registers the method. In the next line, we have used that method and passed the variable “myName” that we have created earlier. Upon pressing enter, it shows the result as “Hello Mayur Ramgir”.

The new Java Shell is great when you are learning about the different aspects of the programming language. You can directly learn about the behavior of an API by simply running different code elements through the JShell. It is present as an API to offer you excellent REPL functionality and bridge the gap that the previous versions had.

10.6.2 Multi-Release JAR Files

Another new feature is that now the JAR format allows use of multiple versions of the same class files to be placed in a single JAR archive file. These class files have different versions for different releases. This is termed as the option of supporting multi-release JAR files within the archive. It allows the use of a single library file with multiple versions and different programming scenarios in the form of Java platform releases.

10.6.3 Platform Support for Legacy Versions

Java 9 has an enhanced javac compiler with the ability to run programs that may have been prepared using the earlier versions of the programming platform. You can employ the -target or the -source options in Java 9, which can cause problems when

compiling for different platforms. However, Java 9 allows the use of release option that ensures that you cannot use APIs that are not supported by the current compiler.

10.6.4 Java Linker

The JEP 220 defines the jlink tool which got introduced in Java 9 that empowers the assembly and the linking process of classes and libraries. In many situations, we don't need an entire JRE to run our programs. For example, to run a simple calculator application which can only performs addition, subtraction, division and multiplication operations. For such applications, we don't need all the classes that come with pre-built JRE which are nearly 4300 plus in numbers. In this case, we can only get the classes which are absolutely needed that can save on memory consumed by runtime. This can be done by JLink. It optimizes modules and ensures that their dependencies can produce a reliable runtime image for functionality. Java Linker is a tool that provides transformation during the assembly and allows the use of different image formats.

10.6.5 Hash Algorithms

Java 9 now supports the powerful SHA-3 hash algorithms that are defined in the NIST FIPS 202. It also supports cryptographic elements that significantly improve the security that Java 9 offers over the previous versions. The java.security.MessageDigest API describes the use of the following hash algorithms as supported by the new platform:

1. SHA3-224
2. SHA3-256
3. SHA3-384
4. SHA3-512

10.6.6 Modular Java Packaging

Java 9 is entirely based on the use of modular packages. This allows for creating a robust programming environment where you can easily create the required programming solution. You can implement a new phase of linking that can separate the execution runtime image phase from the compilation time. This is possible with the jlink tool that we have already described in Section 10.6.4.

There are also options that can be added to specify the position of different modules when using the linker, the compiler, or the JVM. The modular system now also provides access to module-based JAR files. These are the files which you can add in a module-info.class form and put them in the root directory of the platform.

There is now the JMOD format available which is a new package format that provides significant improvements over the currently available JAR format. It can include configuration files as well as native code elements and is activated using the jmod tool.

The JDK is presented in the form of several modules. This allows you to use a variety of configurations, according to your runtime environment needs. The modules can offer compact profile performance as described in the previous Java version. There is also a new scheme which at runtime allows to add new Java elements to the image without affecting its format.

Some redundant tools like java.* , sun.misc, sun.reflect, java.awt.peer, etc. are removed to simplify various JDK APIs and it is now possible to employ various modular options directly from the command prompt by accessing the directory of the installed JDK.

There are five types of modules in the new module system:

- 1. System Modules:** These are the Java SE and JDK modules which can be viewed by list-modules command shown below.

```
java --list-modules
```

Run the above command in a terminal window. Upon running the command we get the following output.

```
Mayurs-MacBook-Pro:~ Mayur$ java --list-modules
java.base@11.0.1
java.compiler@11.0.1
java.datatransfer@11.0.1
java.desktop@11.0.1
java.instrument@11.0.1
java.logging@11.0.1
java.management@11.0.1
java.management.rmi@11.0.1
java.naming@11.0.1
java.net.http@11.0.1
java.prefs@11.0.1
java.rmi@11.0.1
java.scripting@11.0.1
java.se@11.0.1
java.security.jgss@11.0.1
java.security.sasl@11.0.1
java.smartcardio@11.0.1
java.sql@11.0.1
java.sql.rowset@11.0.1
java.transaction.xa@11.0.1
java.xml@11.0.1
java.xml.crypto@11.0.1
jdk.accessibility@11.0.1
jdk.aot@11.0.1
jdk.attach@11.0.1
jdk.charsets@11.0.1
jdk.compiler@11.0.1
jdk.crypto.cryptoki@11.0.1
jdk.crypto.ec@11.0.1
jdk.dynalink@11.0.1
jdk.editpad@11.0.1
jdk.hotspot.agent@11.0.1
jdk.httpserver@11.0.1
jdk.internal.ed@11.0.1
jdk.internal.jvmstat@11.0.1
jdk.internal.le@11.0.1
jdk.internal.opt@11.0.1
jdk.internal.vm.ci@11.0.1
jdk.internal.vm.compiler@11.0.1
jdk.internal.vm.compiler.management@11.0.1
jdk.jartool@11.0.1
jdk.javadoc@11.0.1
jdk.jcmd@11.0.1
jdk.jconsole@11.0.1
jdk.jdeps@11.0.1
jdk.jdi@11.0.1
jdk.jdwp.agent@11.0.1
jdk.jfr@11.0.1
jdk.jlink@11.0.1
jdk.jshell@11.0.1
jdk.jsobject@11.0.1
jdk.jstard@11.0.1
jdk.localedata@11.0.1
jdk.management@11.0.1
jdk.management.agent@11.0.1
jdk.management.jfr@11.0.1
jdk.naming.dns@11.0.1
jdk.naming.rmi@11.0.1
jdk.net@11.0.1
jdk.pack@11.0.1
jdk.rmic@11.0.1
jdk.scripting.nashorn@11.0.1
jdk.scripting.nashorn.shell@11.0.1
jdk.sctp@11.0.1
jdk.security.auth@11.0.1
jdk.security.jgss@11.0.1
jdk.unsupported@11.0.1
jdk.unsupported.desktop@11.0.1
jdk.xml.dom@11.0.1
jdk.zipfs@11.0.1
```

The above list of modules are grouped into four major groups like java, javafx, jdk and Oracle.

- 2. Java modules:** These modules contain the implementation classes of the core SE language specification.
 - JavaFX Modules: These are the FX UI libraries
 - JDK Modules: These are the modules needed by JDK
 - Oracle Modules: These are the oracle specific modules
- 3. Application modules:** These are the modules chosen by us that are needed for the execution of our program. This required list defined in the module-info.class file.
- 4. Automatic modules:** These are the modules which come from the users and not included in the official build. The names of these modules are derived from the name of the jar file. Automatic modules get read only access to the other modules.
- 5. Unnamed module:** These are the modules like classes or JARs which are loaded into the classpath directly but not in the module path.

10.6.6.1 Problems Solved by Modular Java Packaging

There are various benefits of Modular Java Packaging as it tends to solve a lot of problems programmers were facing earlier. Some of these problems are listed below.

- 1. Memory intensive JDK:** JDK comes with a lot of modules which may not be needed for your application need. A lot of memory is wasted on these modules as loading them adds no benefit to your application at all. Here, Modular Java Packaging helps to group the only needed modules.
- 2. Unwanted exposure of modules:** Many times we want to hide or prevent the exposure of some of the modules to the outside modules. We may want to restrict the use of these modules to the internal modules from the same module package. These modules stay hidden or encapsulated. This is what you can achieve with Modular Java Packaging as only the required modules can be exported which will be visible to every other module.
- 3. Runtime failure due to missing module or jar or class:** Prior to Java 9, the missing classes were not detected until the application is actually trying to use them. This was leading to the runtime error. With Java 9 Modular Java Packaging, even Java applications must be packaged as Java modules too. This makes the application to specify all the modules it expected. This enables the Java VM to check the entire module dependency graph for the application. If VM does not find all the dependent modules at start up then it reports the missing modules and shuts down.

10.6.7 Tuning Improvements

Java 9 offers improved tuning improvements. Redundant garbage collection combinations are removed to simplify the use of customized heap recollection. The G1GC that we define in the next subsection is placed as the main GC method, with several tuning parameters placed to improve its performance as the automatic garbage collection choice.

Concurrent Mark Sweep (CMS) now cannot run in the foreground as it is simply not required. The flags that govern the foreground CMS status are now also removed, improving the overall process of performing garbage collection through a reduced, but an empowered set of GC tools. All tools that lost their functionality in the previous version are now completely removed, which increases the efficiency of the current GC method and the available tuning options.

However, the current tuning improvements require programmers to learn more about G1GC, as they will now be using it if they do not want to perform manual tuning functions on a wide scale. The G1GC improves tuning because now it can make better decisions whether to improve memory usage or the processing ability during each cycle of execution.

10.6.8 G1GC as Default Collector

Garbage First Garbage Collector (G1GC) is now the primary GC method, which is automatically activated when programming in Java 9. It is a collector that is designed to provide low-pause collections. This is essential because the

modern Java programming focus is on preparing cloud-enabled applications. These applications must use huge memory heaps while allowing users to suffer from as little downtime as possible when performing an image execution. The G1GC will offer a better stop response for most Java 9 program users, with its ability to alter processing load and memory availability to reduce the collection pauses as much as possible, without causing a problem in the reliability of the execution of the program.

The G1GC is now set up to provide advanced tuning facility. It is possible to change its parameters to ensure that you can use it in the best manner, in the context of your specific program created using Java 9.

10.6.9 Process API Updates

The API process updates provide a lot more power to the Java 9 platform. The ProcessHandle is a new class that now allows programmers to learn about the parameters of the native processes that occur in the programming language. You can get information about the CPU processing time, the parent process, and the descendants that appear from the main process chain.

There is the .onExit method available in this class that allows setting of different mechanisms that can occur when this particular process exits during the runtime execution.

The new API updates allow the developers to obtain the native process ID and then use it to perform functions without turning to the use of native code that may limit the performance of a particular program. The Process class is now full of new methods that are capable of offering the improved insight into a Java stream such as interface:dropWhile, takeWhile, ofNullable, etc.

It is now possible to run the optional functions that offer you more functionality. You can use the `getPid()` method to get the ID of the process which is currently in use. The method now returns the data type (literal) of long type, to allow for larger ID numbers. These new additions empower developers to use better programming aids and ensure that they can employ the information of the available Java processes to create dynamic programs that produce effective results in a real-time environment.

10.6.10 XML Catalog

Java 9 provides a new, standardized XML Catalog in the form of an API. This Java 9 API supports the powerful XML Catalogs that are described by version 1.1 of the Organization for the Advancement of Structured Information Standards (OASIS). This API allows programmers to get the definitions of XML catalogs and can deliver improved performance when using a Java program with the JAXP processors that run the resolver abstraction tools.

The applications that currently use the previous libraries or specific applications to provide the XML support functionality now have to turn their attention towards the new API. This will allow these applications to get more power and gain the ability to employ the new methods and features that are available in the highly improved XML Catalogs.

The XML was an excellent choice for programmers who needed to place data elements on web-based applications. The modern API allows the creation of catalogs to support local Java programs. It is easy to use and ensures that XML processors present in the JDK can use the best features of the available environment to optimize the processing XML based data elements.

The main purpose of this API remains to ensure that the use of external resources is easily implemented within Java programs and does not disturb the program behavior. It improves program performance, enhances the availability of the resources, and significantly improves the security of the platform when used with trusted XML sources.

10.7 | Eclipse IDE for Programming



Eclipse is an excellent IDE that is most widely used by the Java Community. It provides an ideal workplace where you can create your Java projects and work on them in a systematic manner. There is a plug-in system available as well. This allows the addition of external elements that can significantly improve the functionality of the platform.

Eclipse is designed for creating Java applications, but installing different plug-ins allows you to implement common programming languages such as JavaScript, Fortran, C, C++, Ada, PHP, Python, Ruby, and several others. The creation of

packages and documents is possible for the software suite of Mathematica. Common development tools are usually set up for Java and Scala.

You can install Eclipse IDE that provides powerful tools for computer programming and the ability to employ an XML editor and implement the functionality offered by the Maven project management tool. A typical package comes with all the essential tools that you need to start working on your complex Java projects. Here, we will present some basic functions and elements of this programming environment that will allow you to start your journey towards becoming a successful Java 9 programmer.

10.7.1 Advantages of Using Eclipse

As any IDE, eclipse provides a favorable environment for your chosen programming language. Eclipse offers support for languages other than Java as well. However, it is widely used for Java and J2EE related development. Following are some of the benefits of using Eclipse

1. Eclipse is a free and open-source development model and so is constantly updated with community support.
2. Eclipse has large integration options with other tools.
3. Eclipse offers various perspectives for various needs, for example, Java Perspective for Java development, Debug Perspective for debugging purposes, etc.
4. It has built-in Source Control plugins that can connect to various source control tools like Git, Subversion, CVS, etc.
5. It offers various language-related tools such as compilation, execution, debugging, dependencies management, auto completion, etc.

10.7.2 Setting Up Eclipse

Eclipse works on top of your JDK. It offers the abstraction layer that we have defined already as one of the most important principles of OOP programming. It can only run properly when the IDE knows where to find the JDK. This means that once you have installed and opened the software, you should set up the installed JREs from the Preferences>Java>Installed JREs option.

You can simply select the JDKs that you want to use, while uncheck the ones that you do not want Eclipse to identify and implement in its Java abstraction scheme. To add a new JDK, simply identify the relevant directory through a browser window and the software will automatically ask you to confirm your selection if the JDK is found in the described file folder. Please visit Eclipse website to learn more about installation procedure: <https://wiki.eclipse.org/Eclipse/Installation>

10.7.2.1 Starting Eclipse

Java programmers must understand that Eclipse is much more than a simple IDE. It is a development atmosphere that contains the entire set of tools that you will ever need for developing Java projects. You can install the relevant Eclipse package from the website according to your particular computer system. Once you have installed the IDE, it is time to understand its basic setup.

The Eclipse development environment consists of four main sections. These include:

1. **Workspace:** The most important of these elements is the Eclipse workspace. It is the area that has a record of all your current projects.
2. **Perspectives:** The perspective is defined as a particular view of the projects that you have. A single perspective may allow a programmer to use multiple viewing options.
3. **Projects:** The projects contain your actual Java code and the details that you need for programming.

Views: Eclipse IDE provides various views which show project related metadata. For example, the Outline view shows various classes, variables, methods, etc. from the project (Figure 10.15).

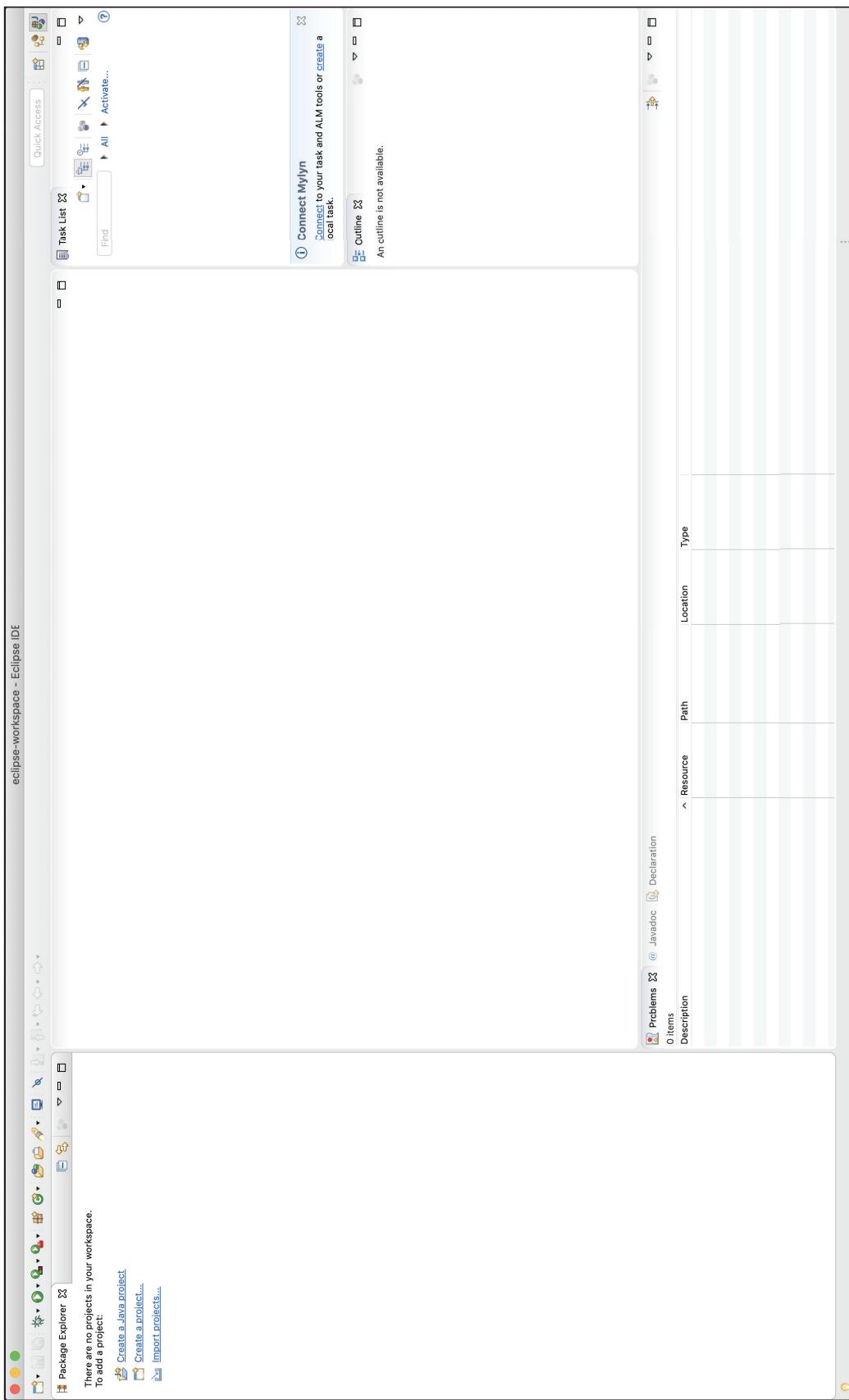


Figure 10.15 Eclipse IDE view.

The screenshot in Figure 10.15 is the example of the perspective that you get when you are viewing your projects in the workspace of an Eclipse window. The perspective provides access to the tools that you need to begin your Java projects. The singular tabs provide different views of the same perspective. The Outline and the Package Explorer are two important views that you will often use in Eclipse.

Remember, all views can be easily docked and moved around according to your convenience. You should take advantage of this flexibility and set up tools as you like them. However, we recommend that you start with the default settings that come with Eclipse.

10.7.2.2 Creating Projects

Creating a new project is easy in Eclipse. Follow Figure 10.16 to create a new project. You need to go to File menu, this will open up menu options. On that slide move your mouse to the New option which will open up a new option pane. On that option pane select Project. This will start a new project, where you can once again verify the specific JDK that you want to use and then click on the Finish button to end the wizard and start working on your project.

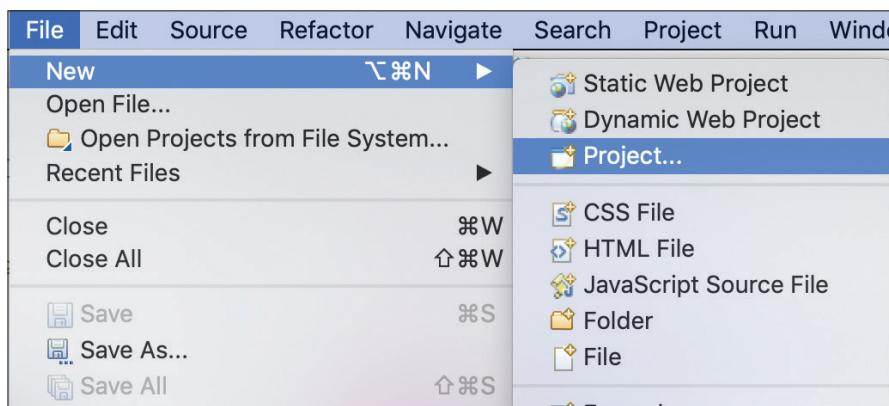


Figure 10.16 Create new project in Eclipse option.

A new Eclipse project automatically creates its source folder and prepares the development environment for immediate use. Understanding the principles of object-oriented programming is then the only thing required to create the necessary Java programs and code elements. Writing code is the next step and is accomplished using the Eclipse Package Explorer.

One of the main elements that we have already described above is the use of the `main()` method that you will see in every program, which can be used for testing as well as defining the required functionality in a Java program. The Logger class is also important when you are learning the use of Eclipse, as it provides information about how the application behaves during its execution, giving you information about how your code is performing.

10.7.2.3 Creating Packages

The Java Package Explorer is an excellent view which is available in the Java Perspective. Eclipse shows you all the information that you need for creating classes. However, you must create packages which are constructs that provide an address for the classes, where they can be mapped and saved according to the available file system on the computer.

The main idea of using Packages is to group the related classes like the way you place files in a folder. Think of a Package as a folder which can contain class files. Before you write your classes it is a good idea to design a good package structure to group the relevant classes. You can place all the relevant Java classes in one package like `com.zonopact.mycoreclasses`. With this structure, IDE will create three folders like “`com`” folder which contains another folder named “`zonopact`” and this “`zonopact`” folder contains “`mycoreclasses`” folder. Now let us look at how to create classes.

10.7.2.4 Defining Classes

Now that you have created a customized package and the source folder to store your classes, it is time to select a class from the New option in the File menu. This will open up the dialog box where you can name your class which will then appear on your

editing window in the Eclipse view. A good way to work on Eclipse, especially if you are a new user, is to close the views that you do not require so that you can easily work on your code. Eclipse remembers your choices and the next time you open the IDE, you have to deal with fewer windows.

Once you generate your class, Eclipse will create the shell for you and other package-related information at the top of your coding window. All you need to do is to now enter more details of your class, such as the variables and the methods that you want to be included as part of your class.

Once you start writing the code within the class, the Eclipse IDE can pick up all syntax errors and show them with underlines. This allows you to find that you have used an undefined word. It means that the final code copy that you create is free from syntax errors and will get compiled. However, the code can always have logical errors in it.

Now that you have initiated your Java class, it is time to prepare the coding elements. Remember, we already described that you must specify the access, the data type, and a unique variable name when defining the attributes of a Java class. The data type is of special importance in this context. It can either be of a primitive type such as int, Boolean, and long, or a non-primitive type such as String and Array, which you will learn in Chapter 11.

Next, you should set up the program logger, which we have described earlier when discussing the basic language tools.

Now we are back to coding the variables that show the attributes of the class. You can define any number of variables within the class easily in the Eclipse IDE. However, you do not have to manually type the getters and setters as you can go to the option of Source as shown in Figure 10.17, by moving the cursor to the class definite. When you select the Generate Getters and Setters button, it adds get and set methods for all the variables in the class. These are important methods to have in order to restrict the access to variables and reduce the exposure to maintain data integrity. Once you click this button, you get access to all the variables that are present under the class definition.

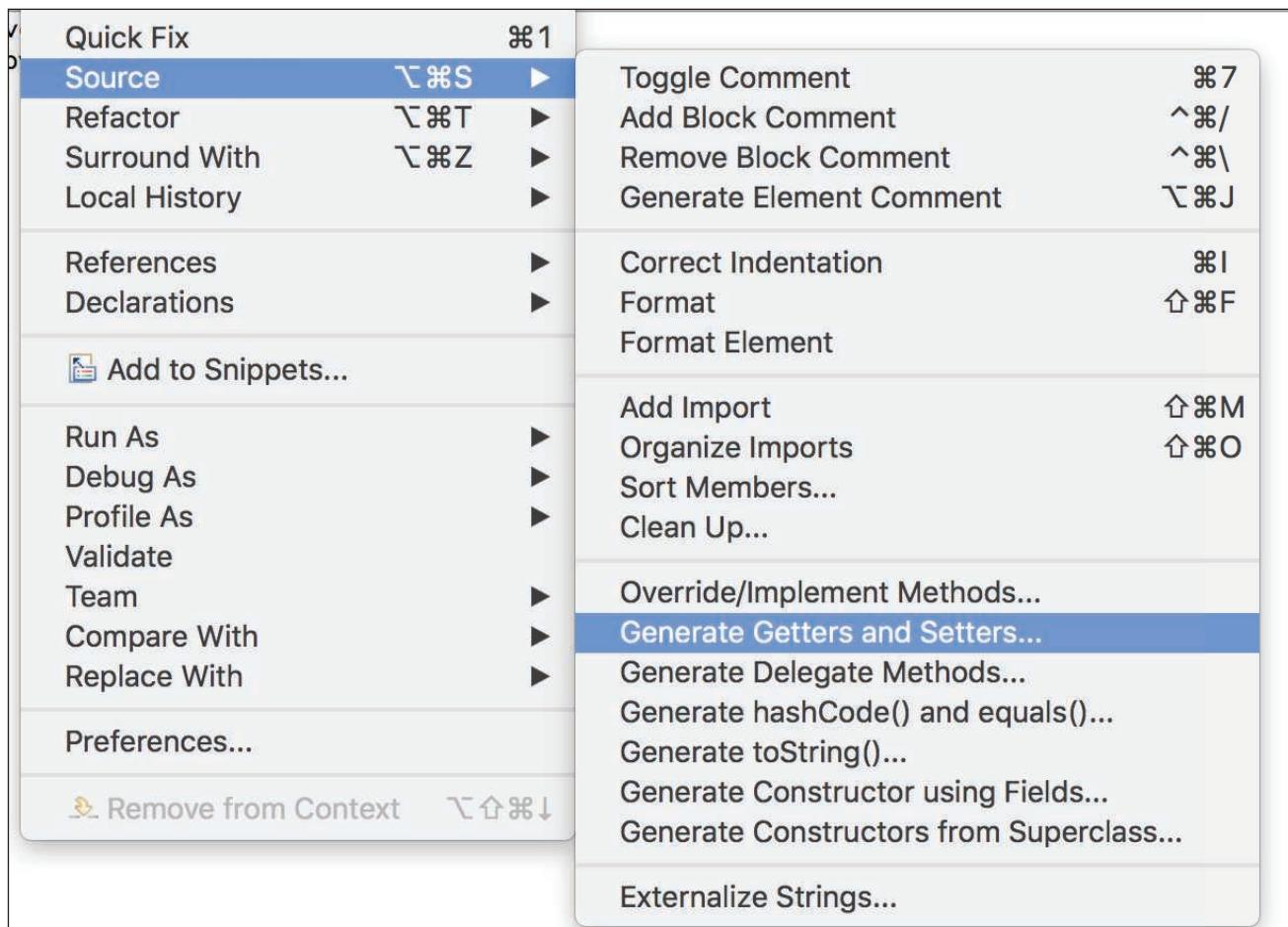


Figure 10.17 Option to Generate Getters and Setters.

An example of using the functionality can be best presented in the following manner:

```
public Book(String name, String genre, int pages, int width, int height, int length,
double cost)
{
    this.name = name;
    this.genre = genre;
    this.pages = pages;
    this.width = width;
    this.length = length;
    this.cost = cost;
}
```

This is a code that you will be able to generate with constructors once the getters and setters are already selected from our described class definition options.

10.7.2.5 Testing in Eclipse

Testing is important as it allows you to ensure that you have set up all the attributes in the right manner and you will not face any problems later on. The JUnit is a testing method that uses your mentioned constructors and then prints the total state of your defined object on the screen console. This allows you to find out that all the constructor instances are properly placed and they are working in the manner you want as a programmer.

Creating a test is easy in Eclipse, where you can simply right-click on your created class and then select the option of JUnit Test Case from New. This will open up the testing wizard of the IDE. Simply select the default options and go ahead by clicking on Finish, which will generate the JUnit test case.

As shown in Figure 10.18, running the test is easy, as you simply need to right click and go to “Run As” and select JUnit Test option. This will produce the status of all the variables that were described in the class using the Logger output.

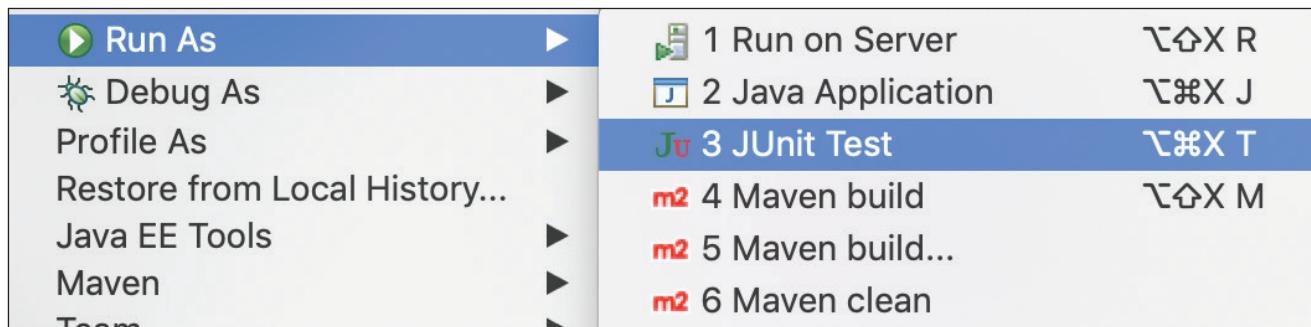


Figure 10.18 JUnit test run option.

10.7.2.6 Java Class Behaviors

One advantage of employing Eclipse IDE is that it allows you to perform several access options directly. The access methods always follow a fixed pattern which is termed as the JavaBeans pattern and is built directly within the IDE. It includes declaring attributes as private, while the getters and setters are always public.

A getter does not take in parameters and returns the same data type as that of the attribute. The setter takes a single parameter which must be of same type as the attribute to which the parameter value is to be set. The setter does not require to return a value.

Using behaviors also requires the calling of methods which is easy to perform in Java. The common way of invoking a method is simple where it follows the following scheme:

```
Object.requiredMethod();
```

The first part provides reference to the object that is used, while the “.” defines that the next word to follow is the method to be employed. The method may take in arguments if required. The required parameters are all mentioned with the parentheses. Methods can be nested just like functions in C, where they can be used within each other for improved functionality. Consider the following example:

```
Logger l1 = Logger.getLogger(LearnJavaProgramming.class.getName());
l1.info("Name: " + p.getName());
```

This is a code where the `getName()` method is employed within the `getLogger()` method to create a nested method design. This functionality is commonly used when creating ordinary programs in Eclipse.

Summary

Java 13 and Java 11 brought some amazing improvements over the previous versions. Moreover, the major updates in Java 9 provide better heap management as well as the ability to reduce the stop-the-clock pauses that may appear during the execution of complex programs. This language is easy to employ if you already have some programming experience. However, if you are new to the language, we suggest that you first familiarize yourself with the principles of object-oriented programming rather than worrying about learning the required syntax and code elements.

In this chapter, we have learned the following concepts:

1. Basic principles of Java programming language.
2. Working of Java Virtual Machine and Compiler.
3. Garbage Collector and how it works.
4. Java Development Kit, Java Runtime Environment, and their difference.
5. What is Object and basic principles of Object-Oriented Programming.
6. Basic idea of Encapsulation, Inheritance, Polymorphism, and Abstraction.
7. How to write program in Java.
8. Basic syntax of Java language.
9. Reserved Words, Variables, and Constructor.
10. New features in Java 9.
11. How to set up Eclipse IDE and use it to write programs.

With this you got the information that you require in order to start programming with Java. We strongly encourage that you experiment with the language to create unique programs that allow you to generate the required functionality. With the use of Eclipse IDE, you will find coding in Java to be an easy affair, where you can gradually move on to create more complex programs as you learn about the various aspects of this excellent programming platform.

In Chapter 11, you will learn the structure of a Java program and also explore concepts such as identifiers, keywords, literals, variables, code blocks, comments, Java packages, primitive classes, static and non-static methods, string options, arrays, enums, overriding, autoboxing, unboxing, various types of operators, expressions, control flow, loops, etc.

Multiple-Choice Questions

1. Which of the following tools is utilized for compiling Java code?
 - (a) Jar
 - (b) Java
 - (c) Javadoc
 - (d) Javac
2. Which of the following tools is used to execute Java code?
 - (a) Javac
 - (b) Javadoc
3. (c) rmic
(d) Java
3. Which tool is available to generate API documentation in HTML format from Java source code?
 - (a) Javadoc
 - (b) Javamanual
 - (c) Javahelp
 - (d) None of the above

4. What is the full form of jar?
- Java Archive
 - Java Archive Runner
 - Java Application Runner
 - None of these
5. Which of the following is not known as a keyword in Java?
- Assert
 - Boolean
 - Abstract
 - Finalize

Review Questions

- What is the difference between JDK and JRE?
- How is JVM useful in executing Java code?
- What is Garbage Collection and how does it work?
- How do you define Object and Class?
- What is Object-Oriented Programming?
- List Object-Oriented Programming principals.
- How is inheritance useful?
- What is package and how do you create one?
- What is access modifier? Name and describe all of them.
- What is the default access modifier?
- What is reserved word in Java? Name at least 10 and explain in detail.
- What is the difference between constructor and method?
- What is the use of constructor?
- Can method be private?
- What is variable? Give some examples.
- What are some of the prominent features of Java 9?
- How can one set up Eclipse and create a new project?
- Can you test a project in Eclipse?

Exercises

- Think of a real-life example of inheritance, polymorphism, abstraction, and encapsulation. Design class structure for these examples.
- Draw a flow chart to explain the flow of Java Code from plain text to executable.
- Create a chart to explain the benefits of using Java 9 over Java 8.
- Using Eclipse IDE, create a simple project, add one package and 3 classes.

Project Idea

Take an example of a vending machine that dispenses food items upon accepting money. State the object-oriented principal that this vending machine is based on and explain the entities and their relationships.

Recommended Readings

- Oracle Java Doc: <https://docs.oracle.com/javase/tutorial/>
- W3schools: <https://www.w3schools.com/java/>

Language Syntax and Elements of Language

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Basic syntax structure to create simple Java programs.
- Primitive data types and how they can be employed in the Java syntax.
- Keywords which are specified in Java and cannot be used to denote the names of methods and data instances.
- String options, which play a vital role in the syntax of Java and how we can use its class structure.
- Various ways in which programmers can use classes, objects, and methods.
- Concept of implementing logic in Java programs by creating relevant classes and following the ideal OOP principles.
- Wrapper classes, autoboxing, unboxing, etc.
- Various operators.

In Chapter 10, you were introduced to some of the concepts of Java. We only tried to give you some basic idea of Java, program structure, syntax, etc. In this chapter, we will explain these concepts more thoroughly along with examples.

Syntax is a linguistic term, which defines the set of rules that govern how a language can be employed. It includes the order of words, their structure, and suggests how they deliver information to the language users. If we think about programming languages, their syntax consists of the basic rules that govern the program development in their specific environments.

The goal of this chapter is to familiarize you with the Java language syntax. We will describe the various components that make up this language and suggest the important elements that you need to learn to become a Java programmer.

11.1 | Building Blocks of Java

Java is an object-oriented programming (OOP) language. It is a high-level language, which encourages developers to use high-end logic and focus on creating complex functions, rather than limiting their focus on understanding and developing the basic language tools for their programs. As we have defined syntax in the linguistics, the Java syntax is similarly a set of rules that define how to write code for the Java virtual machine (JVM) compilation.

Java derives its basic structure and element names from C and C++ programming languages. Although it has a different working structure from these languages, because of the absence of global elements, it still uses the ease of common English words that represent functionality on offer. The language employs the structure of classes.

All Java code is defined in terms of classes, which may often form part of a library. All instances of these classes are termed as *objects*. Java is slightly different from a pure OOP language since it has primitive data types that are present within the language to ensure improved performance. There are some features which are possible in Java but are restricted to ensure that Java programmers cannot make mistakes. Such mistakes are difficult to identify and resolve, especially in large and complex programs.

Here, we first represent a basic Java code element and then describe how it is structured to give an idea of the syntax of this programming language:



```

public class MyVeryFirstJavaProgram {
    public static void main(String[] args) {
        // This code will present the message within the quotations ahead in the terminal
        // display window.
        System.out.println("First Java Program");
    }
}

```

This is a simple example. The first word is “public” is a keyword in Java. These are words which have special meaning. The second word is another special word of “class”. Remember, all Java programs consist of classes and class objects. Think of a class, as a standalone definition, which provides a structure but does not have a body of its own. The class name here is `MyVeryFirstJavaProgram`, which is termed as *identifier*. An identifier is a unique word that points to a particular class, object, or a particular method.

The keyword “static” describes a specific meaning in the syntax. Then we find the `main()` method. This is a method which must be present in your Java program for it to actually execute something. This is always run first by the program and can contain various program objects, which may be described with the use of different classes and the creation of a hierarchy.

There are also some words in parentheses. These are words that work as arguments to the methods, which are employed in Java programs. The line that is inserted with two forward slash symbols is the comments line, where we write what this code section does. Comments are extremely useful, and programmers employ them all the time to gain useful information about various code elements. This is especially true when multiple people work on a single Java project, where it is important to know what each program section does to avoid problems.

The next line presents the use of a method. Remember, methods in Java are code elements that perform the actual tasks. They represent the implementation of various schemes and are often used from the inherent Java libraries which become available with Java Runtime Environment (JRE), which is installed on all systems that are programmed to run Java programs using JVM instances.

Now, we look at how the code works. `System` is a Java class that contains various objects and methods that offer console control to Java programmers. The method of `println()` prints the string message contained within the quotations and then returns the console cursor to a new line. This is an improvement over the traditional console outputs of other languages, where you must manually program a movement to the new line.

Remember, method instances are finished using the semicolon sign “;”. Another way to create the structure of any method, class object, or other values in Java is to employ curly braces “{}”. They are used to employ compounded statements, where you want multiple independent code lines as a single coding structure.

These are some basic elements that formulate a Java structure. Remember, Java recognizes the end of a statement with a semicolon. We only use spaces and moving to a new line as tools to help us organize the programming project in a visual manner. Program comments are also not a necessity for use in Java. However, we use them to create an effective programming code where any programmer can perform the required analysis and study the code to make further additions or required improvements.

You can write the program that we have described in any text editor environment. Although there are some excellent Java programming platforms like Maven, all programs can be created using any ASCII-based editor. This program can only be compiled when you save the program with an extension of `.java`. Remember, your filename should be the name of your main program class. In this case, our program will be named “`MyVeryFirstJavaProgram.java`”, where it is perfectly ready to be compiled using a Java installation.

The Java compiler can then be run on the final program, where it will create a compiled file if the code elements are correct and do not produce a compiler error. This file is then prepared to be executed if a system has a Java environment installed on it. This process describes the entire structure of a Java program and how it achieves the required maturity to perform the intended functionality on a computer system.

11.1.1 Keywords

A keyword in a programming language is a reserved word that can only be employed in writing code to represent a specific scenario. There are about 50 keywords in Java. They have their specific meaning and represent special functions. All integrated development environments (IDEs) use a special highlighting scheme to mark such words. This makes it easier to read the Java syntax and find out how the code is arranged and supposed to function.

These keywords often function as providing the additional performance over the OOP potential. They especially include access modifiers and specific words that explain the basic functionality and describe how class objects will function. They also include other tasks such as memory handling and the intrinsic options which are available in the programming language. We have already covered this section with the list of keywords in Section 10.4.7 of Chapter 10.



Can you use these keywords as variable names? If yes, give an example. If not, can you use these keywords with case change? For example, class is a keyword so you can instead use class as variable name.

11.1.2 Primitive Classes

As we have already discussed, there is still a need to include basic values, even in OOP languages. There are eight primitive classes, or ones that are often termed as *primitive wrapper classes*, which provide an identity to the common types of data that we must employ. Boolean represents a class, which describes the results of a decision-making element. The types of byte, short, int, and long describe integers. The types of float and double represent accurate decimal numbers. The char type represents a single character value. We will be covering this in more details in Chapter 17. For now, let us try to understand this concept. Listed below are the various data types.

- **boolean:** It describes two values of true and false, with the default option set at false. It can be used to record the results of decision-making.
- **byte:** An 8-bit signed integer data type which is described as 2's complement. It has allowed values from -128 to 127 and is excellent for use in limiting data storage in large arrays. It has a default value of 0.
- **char:** It is for recording Unicode characters that are stored as 16-bit single character. Its values start from \u0000 to \uffff, which translate as 0 to 65,535 in numbers. The default value is set to \u0000.
- **short:** It is a data type that allows for 16-bit storage of signed numbers that range from -32768 to 32767. It offers greater range than a byte and has the same purpose of allowing the use of reduced memory in large arrays. The default value is 0.
- **int:** This is the primary integer data type that uses 32-bits, which originally allowed for storing signed integers ranging from -2³¹ to 2³¹-1. However, Java 8 and later, it allows the use of unsigned integers, which gives the range as 0 to 2³²-1. It stores numbers using the form of 2's complement. It is the primary integer primitive data type commonly employed in Java programs. The default value is 0.
- **long:** Also representing integers, it uses 64-bits where the numbers are stored using the 2's complement form, a convenient method for storing integers. The signed digits range from -2⁶³ to 2⁶³-1, while the unsigned integers present from Java 8 onwards can have values from 0 to 2⁶⁴-1. The default value is 0.
- **float:** This data type represents a floating-point number with single precision using IEEE 754 32-bit standard. The float is often used to represent floating numbers that not require a high degree of precisions. It is excellent for storing data in arrays. The default value remains 0.0f.
- **double:** This is a double precision floating point value storage data type. It uses 64-bit storage and offers a much higher range of accuracy, which is perfect for use in applications such as currency conversions. According to the coding use, you should always double to store your precision numbers.

Table 11.1 compares the values for the primitives.

Table 11.1 Primitive values comparison

Type	Bits	Bytes	Minimum Range	Maximum Range
byte	8	1	-2 ⁷	2 ⁷ -1
short	16	2	-2 ¹⁵	2 ¹⁵ -1
int	32	4	-2 ³¹	2 ³¹ -1
long	64	8	-2 ⁶³	2 ⁶³ -1
float	32	4	1.4E-45	3.4028235E38
double	64	8	4.9E-324	1.7976931348623157E308

Any value which is stored using these primitive data types does not have to be defined in a class and is termed as *literal* in Java.



Can int hold a value of 10.3?

11.1.3 Literals

Although Java is an OOP language, it still provides specific functionality by providing data literals. These are intrinsic data types that are simply too important to include in any programming language. They represent the specific instances of the primitive data types, including options such as numeric, Boolean, character, and string data representations. You express particular data value cases using these literals.

These literals allow us to represent various number schemes such as binary, octal, hexadecimal, and decimal notations. It also allows the use of floating point data instances while two Boolean values of *true* and *false* are also available. There are string literals that are represented by writing within a set of double quotations. The language also provides several character escape sequences that are triggered with the use of a backslash “\”.

You can view the entire list for Java 11 by going through this related Oracle page: <https://docs.oracle.com/javase/specs/jls/se11/html/jls-3.html#jls-3.10>

11.1.4 Variables

Variables identify specific literal values by using unique names that have already been explained. The first word is a keyword, which describes the type of variable such as int, float, and String. The unique name follows. You can initialize the variable with a value when it is declared or simply leave the declaration and it is initialized later. The following are a few examples of variable instances:

1. int num;
2. num=0;
3. int num2=5;
4. int num1=2, num4=6;

These examples show how it is possible to use the variables in different ways. You can define multiple variables in a single notation, by separating each instance using the delimiter of a comma “,”.

11.1.5 Code Blocks

As already discussed, we use curly braces to create blocks of code. These blocks act as singular elements to their attached objects, methods, and any other units of coding in Java. You will find that each program has several instances of these code blocks which represent some specific functionality. Given below is an example showing the use of various blocks::

```
void new Method() {
    int firstnum;
    {
        int secondnum;
        firstnum=1;
        System.out.println("first number is "+firstnum);
    }
    firstnum=2;
    System.out.println("first number is now "+firstnum);
    secondnum=5;
    // This is an illegal use of variable identity, since a variable declared in a code
    block will not remain available as a universal variable outside of that particular
    code block
}
```

This is an excellent code block example, as it shows that we can use multiple nested code blocks. However, they all represent the quality of presenting local and global values. The local variable of second num will not work outside of its code block. However, if we create further nested loops within its block, it will continue to work with the branched blocks in a normal way.

QUICK CHALLENGE

Think of a use case in which you will utilize the code blocks.

11.1.6 Comments

We have seen the additional elements in Java other than code. They are the comments that provide guidance to other programmers as well as yourself, so that you can easily review your code when you want. You can use traditional comments with which you may be familiar from other languages such as C and C++. These are comments that start with “`/*`” and end with “`*/`”.

You can employ them in any number of lines, as moving to new line is not recognized by the compiler. You can also use the end of the line comments, which are initiated with the use of “`//`” symbol (just as in the example given above). This comment continues till the line is eliminated by moving to the next line. They are useful for providing the exact functionality of the particular code elements, and is a tool often employed in good program coding practices.

There is also a third way of using comments. These are termed as *documentation comments*, as they are used to add information above class, methods on the code lines which can be read as documentation for the code. The *javadoc* (<https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html>) document generator reads these comments and creates the necessary records. Given below is an instance of the use of these comments:

```
/**  
 *This is first line of documentation  
 *This is second line.  
 .....  
 *The nth required line.  
 */
```

This represents a block of documentation which is perfect for inserting in specific sections of your program to define its logic and functional use for other programmers.

11.2 | Calling the Main Method

A key part of the Java syntax structure is the main method. All applications prepared in Java must have an entry point to start the program in a physical environment. This is regardless of whether the application runs in the background on a console setting or provides an interactive graphical user interface (GUI). This is achieved by calling the `main()` method in Java.

It is a static method. We will discuss more about such methods in Section 11.2.1. However, what you must know about the main method is that it can hold multiple classes as they may be required for Java application. However, there is always the ideal use of an external class, which must only hold the main method and provide the instance for naming the Java file. This method in Java does not return a value and has the intrinsic design of always employing “void” to describe its return type.

11.2.1 Static and Non-Static Methods

All methods that are employed in Java always lie under the hierachal structures of classes and objects. However, they can all be divided into two basic types:

- Static method:** This method is defined with the declaration of a class and is therefore available for use throughout any class instances. It may also be employed by mentioning the class followed by a dot to use the method infrastructure in a Java program. Static methods are usually publicly available for use throughout the program.

However, this situation creates a problem because only a single instance of these methods is possible. You cannot use them with multiple copies. Take the example of the `main()` method, which we know will only have a single instance. This makes it a suitable static method in Java programs. Remember, static methods can be used by classes and by any object of the class. However, they can only include members in their structure, which are also static in their nature.

Static methods are uncommon and are not used often in a program. They have limited use, which is important considering that they are designed for a single-use situation. We once again stress that you do not need any object (instance) of a class to call one of its static methods. Let us look at the following example:

```
class DemoStatic {
    public static void copyVal(String st1, String st2) {
        st2=st1; //performs the copying of the first value to the second value
        System.out.println("The First value is: "+st1);
        System.out.println("The Second value is: "+st2);
    }
    public static void main(String[] args) {
        copyVal("ABC", "XYZ");
    }
}
```

We can clearly observe in this example that we have not created any object and have directly employed two static methods. We have first declared a static method `copyVal()` that copies one string argument to its other string argument. We then run the static method of `main()`, which runs the static method with two arguments. Our code changes the second string to have the first-string value. We get a console message of two lines which will both show the values of ABC. These methods are allocated in memory only once, whenever the compiler initiates the class.

- 2. Non-static method:** Most methods that we use in Java are non-static methods. It means that they are employed within an object of a class. They provide the normal functional elements, which carry out the designed logic and create programming solutions in our code. These are the regular methods that we use in our programs. They can have their own memory instances each time they are called in our programming elements. Given below is a simple example of using non-static methods in Java programs:

```
public class DemoNonstatic {
    public void dispfunc() {
        System.out.println("Display using non-static method");
    }
    public static void main(String[] args) {
        DemoNonstatic obj=new DemoNonstatic();
        obj.dispfunc(); //directly using the dispfunc() will cause an error here
    }
}
```

Now, this prints the line in the quotations on the console. However, it is employed only when the newly created object of the `DemoNonstatic` class is created and calls this particular method. A new memory instance will be created for the use of method in such instances. Always remember that non-static methods are always called in the manner of `Object.methodname()` format.

11.3 | String Options

String entries are extremely important in the syntax of Java. These are several string options that are available in the language. String in Java is always initiated as a special class that presents all string data values as its instances. Remember, strings represent constant values in Java; therefore, they are only employed in a specific manner. Each change in value in actuality produces a new instance of String.



The String holds important value in Java since it is capable of representing any of the primitive wrapper classes that represent data values. When employed with the String class, other data values simply get converted to their string versions. The string can still be understood as an array of characters which can then provide the functionality of offering a fixed String instance. There are other important functions as well. Here, we present StringBuilder for this case.

The StringBuilder class objects are similar to normal String, but the main difference is that they can be modified. The class internally behaves as an array of characters, where different methods can be employed to change the content and the length of the stored sequence of characters. The StringBuilder Constructor has the ability to create a set of 16 elements. See the example below to understand how this class works:

```
...
StringBuilder sbr = new StringBuilder();
sbr.append("Greetings"); //adds 9 characters to the array
...
```

This code creates a new `StringBuilder` object and places 9 characters on the first 9 places in the total set of 16 elements. They occupy positions from 0 to 8 in their character array, within the internal workings of the class. It is an excellent class when you want to modify your strings and run operations that may alter its length and perform mathematical functions with some logic. It uses two special methods of `append()` and `insert()`, which give this class its edge and allows it to accept data from any primitive type.

Another related class is the “`StringBuffer`” class in Java. It can also be modified but has the additional capability of allowing for the use of concurrency. Java 9 and the subsequent versions are already going to emphasize a lot on using parallelism in Java programs and applications, where this class certainly has an improved application.



What will be the output of multiplication of two String variables, String `variable1` holds a value of 10 and String `variable2` holds a value of 20?

All methods of this class are already synchronized and work in an overloaded manner when accepting any data. `StringBuffer` usually employs the same two methods of `append()` and `insert()` where both methods work the same for a new object instance. This class is different from the `StringBuilder` because it can accept any capacity. The overflow of the available buffer for this class automatically increases the available limit of the characters that can be stored in an object.

However, this makes the results difficult to control. You can use the `StringBuilder` class objects for swift operations. However, if you are creating a concurrent program where you want to ensure that you implement synchronization, `StringBuffer` objects will certainly provide you better control over the use of code application in a parallel manner.

QUICK CHALLENGE

Create a comparison chart for `StringBuffer` and `StringBuilder`.



11.4 | Arrays

Arrays are an important part of Java as they are required for implementing most types of programs. They are produced as dynamic objects and are termed as the instances of the `Object` class. Arrays can be termed as container objects, where they can hold a fixed number of values belonging to a primitive class. However, it is also possible to declare an array of arrays. We will be covering arrays in more detail in Chapter 17. Arrays are declared using two parts:

1. The first part describes the primitive type of data that the array will hold along with braced “[]” that indicate that this variable is an array.
2. The second element is the unique identifier, which is named according to the policies that we have described above.

Declaring an array simply creates a reference, where it cannot be used without having values. Using the “new” keyword as shown in the example below instantiates it according to the number of elements that we mention.

All values present in an array are termed as individual elements and are identified by a numerical index value. The first array location starts from “0” and then keeps moving forward according to the number of stored values. Let us look at how we use arrays in Java syntax:

```

package java11.fundamentals.chapter11;
public class DemoArray {
    public static void main(String[] args) {
        int[] oneArray;
        // we initiate an integer array
        oneArray = new int[5];
        //We set up 5 memory locations for the array
        oneArray[0] = 122;
        oneArray[1] = 212;
        oneArray[2] = 58;
        oneArray[3] = 125;
        oneArray[4] = 200;
        //We store 5 separate integers in successive positions
        System.out.println("The first element is: " + oneArray[0]);
        System.out.println("The second element is: " + oneArray[1]);
        System.out.println("The third element is: " + oneArray[2]);
        System.out.println("The fourth element is: " + oneArray[3]);
        System.out.println("The fifth element is: " + oneArray[4]);
    }
}

```

The output from this program is simple. It will print out the following lines on the console.

The first element is: 122
The second element is: 212
The third element is: 58
The fourth element is: 125
The fifth element is: 200

As you can observe, it is an impractical approach to enter each value separately, especially when using large arrays. This is to just show an example of the syntax, as most times you will be running iteration loops that increment the array index positions to save values either directly or by using user input in interactive programs.

You can also create an array of all possible data types such as byte, short, long, float, double, boolean, char, and even Strings, which is kind of an array of arrays in itself. It is possible to place the brackets after the identifier, but the standard practice we will follow in this book is to always place them with the type of data. This means to always employ the practice of naming like `long[] largeNum` rather than `long largeNum[]`, although both are valid for the compiler.

An easy method for placing values in a single step is also possible by using the following convention with the above example:

```
int[] oneArray = {122, 212, 58, 125, 200};
```

This will automatically set up an array with five elements, as shown by five values in the brackets, which must be separated by using commas. Java also allows the creation of multidimensional arrays. All string arrays are in fact multidimensional examples of character arrays. However, the difference in Java is that the components in such arrays are true arrays. This means that that array rows in Java can have different lengths. Given below is an example to show this property:

```

package java11.fundamentals.chapter11;
public class DemoArrayMulti {
    public static void main(String[] args) {
        String[][] salName = { { "Mr. ", "Mrs. ", "Ms. " }, { "Alan", "Janice" } };
        // We have elements for only two rows out of three
        System.out.println("The first combined value is: " + salName[0][0] + salName[1][0]);
        // Prints first name combination
        System.out.println("The second combined value is: " + salName[0][2] + salName[1][1]);
    }
}

```

The output for this program shows two lines.

```
The first combined value is: Mr. Alan
The second combined value is: Ms. Janice
```

This example shows how we can use different number of elements in the arrays, as they act as fully functional arrays when employed in multiple dimensions. There are useful functions and methods that you can perform on arrays, such as writing `.length` to return the length of any given array. You can use the method of `arraycopy()`, which copies the contents of one array from the mentioned starting position to a destination array position. Given below is an example:

```
package java11.fundamentals.chapter11;
public class CopyArrayExam {
    static char[] copySource = { 'f', 'g', 'o', 'd', 'a', 'm', 'n' };
    static char[] copyDest = new char[3];
    public static void main(String args[]) {
        System.arraycopy(copySource, 1, copyDest, 0, 3);
        // Copy from first array location by reading from the second element
        // and placing in the second array from the first element, with a length
        // of three elements
        System.out.println(new String(copyDest));
    }
}
```

This program will print “god” on the console.

The copying process is simple to understand. The first argument in the method marks the source array, the second describes the starting position for copying array elements, the third describes the destination array, the fourth describes the starting position for pasting values, and the last position declares the length of the array to be copied.

There are other useful methods available in the `java.util.Arrays` class which are quite useful. They include methods such as `fill()`, `copyOfRange()`, `equals()` and various sorting methods. Java offers the powerful method of `parallelSort()`, which is faster by employing concurrency available of multiprocessor systems commonly available.

11.5 | Enums



Enums is another facility in Java that describe a special type of data which can define a collection of constant values in a program. There may be methods and other facilities which have been present in various Java versions. Enums allow programmers to be dynamic and set up any logic by creating a set of organized constants for use in a program.

Take the example of directions, where there are four fixed values of north, east, south, and west. Similarly, the seven days of the week can form an enum as well, where these values are consistent and would never require a change during program executions. A typical instantiation in Java will occur in the following manner:

```
package java11.fundamentals.chapter11;
public enum CarTypes {Sport, Sedan, Hatchback, SUV, Mini, Hybrid}
```

Given below is an example that will help you understand how to use enum values in a Java syntax and how it is valuable to add functionality in your program by using the enum that was set up earlier:

```

package java11.fundamentals.chapter11;
public class EnumExample {
    CarTypes carTypes;
    public EnumExample(CarTypes carTypes) {
        this.carTypes = carTypes;
    }
    public void carFeatures() {
        switch (carTypes) {
            case Sport:
                System.out.println("Stylish car with power");
                break;
            case Hybrid:
                System.out.println("Economical as partially runs on battery power");
                break;
            case Hatchback:
            case SUV:
                System.out.println("Rear door swings upward to provide access to the cargo
area");
                break;
            default:
                System.out.println("Just a car");
                break;
        }
    }
    public static void main(String[] args) {
        EnumExample carOne = new EnumExample(CarTypes.Sport);
        carOne.carFeatures();
        EnumExample carTwo = new EnumExample(CarTypes.Hatchback);
        carTwo.carFeatures();
        EnumExample carThree = new EnumExample(CarTypes.Mini);
        carThree.carFeatures();
    }
}

```

This is a code which provides an excellent use of enums, while also describing situations where we can implement case and break scenarios that Java brings in from C and C++ languages. This program will print three lines with three created objects that all run the single defined method of `carFeatures()`. Given below is the output.

Stylish car with power
Rear door swings upward to provide access to the cargo area
Just a car

All created enums are the extensions of `java.lang.Enum`, which means that there cannot be a further inheritance of state. Creating an enum will not extend anything other than the single instance of the set of fixed values. There are plenty of ways in which you can use these enums. Remember to always treat them as a constant set of values, and you will be able to employ them in a variety of application settings when you are using the Java syntax for program developments.

Enums offer excellent advantages, and therefore you should employ them in your program syntax wherever applicable. Listed below are some important benefits of using enums for extending a set of constant values:

1. Enums offer excellent safety when you want to ensure that the correct type of data will be available in a particular variable.
2. It is possible to traverse your Java enums providing excellent functionality.
3. As shown in the example, enum objects are perfect for use in “switch” case representations where you want to implement specific responses in a program, according to a specific but fixed set of situations.
4. It is possible to have various constructors, methods, and fields that represent an enum.
5. Enums are perfect for implementing interfaces. However, they do not perform extensions and therefore, offer programming safety.

Enums provide an excellent set of internal methods that are always available once an enum has been set up in a program. They provide excellent additional functionality such as the ability to return the enum values.

QUICK CHALLENGE

Write real life use cases where you will need enums.



11.6 | Wrapper Classes

Java provides various wrapper classes that are useful in the following ways:

- Wrap primitive values in an object:** As you know, primitives are not objects and hence they cannot be directly included in the activities that are reserved for objects, activities like adding itself to Collections. Collections can only take objects and hence primitives cannot get added.
- Utility functions for primitives:** Various functions are not available to primitives such as converting to and from String objects, converting to and from binary, octal and hexadecimal.

11.6.1 Outline of Wrapper Classes

Wrapper class, as name suggests, wraps a primitive in an object. Each primitive has a corresponding wrapper class. For example, int primitive has Integer wrapper class, float primitive has Float wrapper class, etc. Table 11.2 shows primitives and their equivalent wrapper classes.

Table 11.2 Primitives and their equivalent wrapper classes

Primitive	Wrapper Class	Constructor Argument
byte	Byte	byte or String
int	Integer	int or String
short	Short	short or String
long	Long	long or String
float	Float	float, double, or String
double	Double	double or String
char	Character	char
boolean	Boolean	boolean or String

11.6.2 Creation of Wrapper Objects

Wrapper objects can be created in three different ways: Constructor, valueOf(String val), and valueOf(String val, int radix).

- Constructor:** There are two types of Constructors other than Character Wrapper. One takes a primitive value and other takes a String representation of primitive. See the following examples to understand it better.

Primitive examples are as follows:

- `Integer i = new Integer(10);`
- `Float f = new Float(5.5f);`
- `Boolean b = new Boolean(true);`

String representation examples are as follows:

- `Integer i = new Integer("10");`
- `Float f = new Float("5.5f");`
- `Character c = new Character('c');`
- `Boolean b = new Boolean("true");`

Please note that in String representation, Character only provides one constructor which accepts char as an argument.

2. **valueOf(String val):** This type accepts one argument as String. For example:

```
Float f = Float.valueOf("5.5f");
```

3. **valueOf(String val, int radix):** This type accepts two arguments. One is String and other one is radix. For example;

```
Integer i = Integer.valueOf("1010", 2);
```

11.6.3 Wrapper Conversion Utility Methods

One of the ways in which Wrapper is useful is the utility methods to convert primitives. There is a common way of converting to a primitive. We can use the `xxxValue()` format, where `xxx` can be changed to the desired primitive type. Table 11.3 shows some common conversion methods to convert String to a primitive.

Table 11.3 Common conversion methods to convert String to a primitive

	Boolean	Byte	Character	Double	Float	Integer	Long	Short
byteValue		x		x	x	x	x	x
doubleValue		x		x	x	x	x	x
floatValue		x		x	x	x	x	x
intValue		x		x	x	x	x	x
longValue		x		x	x	x	x	x
shortValue		x		x	x	x	x	x

Let us try to understand this with the help of following examples:

1. Create a wrapper object:

```
Integer i = new Integer(42);
```

2. Convert Integer *i* value to byte:

```
byte b = i.byteValue();
```

3. Convert Integer *i* value to short:

```
short s = i.shortValue();
```

4. Convert Integer *i* value to double:

```
double d = i.doubleValue();
```

Table 11.4 shows common conversion methods to convert Wrapper to a primitive.

Table 11.4 Common conversion methods to convert Wrapper to a primitive

		Boolean	Byte	Character	Double	Float	Integer	Long	Short
parseXxx	Static Method Throws NumberFormatException		x		x	x	x	x	x
parseXxx (with radix)	Static Method Throws NumberFormatException		x			x	x	x	x

Let us try to understand with the help of the following examples:

1. `double d = Double.parseDouble("5.5");`
2. `long l = Long.parseLong("101010", 2);`

Table 11.5 shows common conversion methods to convert String to a Wrapper.

Table 11.5 Common conversion methods to convert String to a Wrapper

		Boolean	Byte	Character	Double	Float	Integer	Long	Short
valueOf	Static Method Throws NumberFormatException		x	x		x	x	x	x
valueOf (with radix)	Static Method Throws NumberFormatException			x				x	x

Let us try to understand with the help of the following examples:

1. `Double d = Double.valueOf("5.5");`
2. `Long l = Long.valueOf("101010", 2);`

Table 11.6 shows common conversion methods to give String representation.

Table 11.6 Common conversion methods to give String representation

		Boolean	Byte	Character	Double	Float	Integer	Long	Short
toString		x	x	x	x	x	x	x	x
toString (primitive)	Static Method	x	x	x	x	x	x	x	x
toString (primitive, radix)	Static Method					x		x	

Let us try to understand with the help of the following example:

```
Double d = new Double ("5.5");
System.out.println("d = "+ d.toString() );
```

The above code prints the following result:

```
d = 5.5
```

11.7 | Autoboxing and Unboxing

We have already discussed that the Java compiler is capable of moving elements between different types of primitive datasets. After Java 5, Java compiler is capable of automatically converting the primitive types into their object Wrapper classes. As shown in Figure 11.1, a process of compiler converting from primitive datatype to Wrapper class called *autoboxing* and from Wrapper class to primitive datatype is called *unboxing*.

In other words, we can say that, the compiler is capable of creating these Wrapper classes and this ability is termed as autoboxing. If this option was not present in the operating language, there would be an error every time a method required an object that should represent a Wrapper class in Java. For example,

```
Character ch = 'X';
```



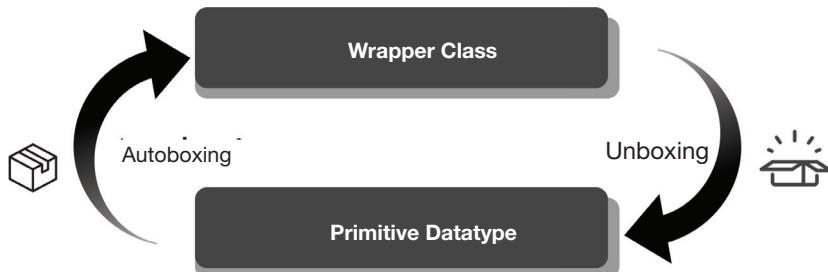


Figure 11.1 Autoboxing and unboxing process diagram.

This explains that there is a primitive data of character type which gets changed into a Character object using the generics syntax. Here is another way in which this functionality is achieved:

```

package java11.fundamentals.chapter11;
import java.util.ArrayList;
import java.util.List;
public class AutoboxingExample {
    public static void main(String args[]){
        List<Integer> alist = new ArrayList<>();
        for (int i = 1; i < 10; i += 1){
            alist.add(i);
        }
    }
}

```

This code reads the primitive integer values and then converts them to Integer objects to ensure that the List can be maintained which is a collection of objects. There are two ways in which primitive to Wrapper auto conversion takes place. The first way is when a value is passed as a parameter where the method definition requires a Wrapper class object. The second way is when a value is assigned to a variable of a specific Wrapper class.

The Java compiler is capable of doing the opposite of this function as well. It can take the example of a Wrapper class object and convert it into primitive data types because they are required to run the required arithmetic operators. This process of creating data from an object is the reverse of the earlier process and is termed as unboxing. Let us take a look at an unboxing example,

```

package java11.fundamentals.chapter11;
import java.util.ArrayList;
import java.util.List;
public class UnboxingExample {
    public static void main(String args[]){
        ArrayList<Integer> nums = new ArrayList<Integer>();
        nums.add(1);
        nums.add(15);
        nums.add(20);
        System.out.println("Total is " + addNumbers(nums));
    }
    public static int addNumbers(List<Integer> nums) {
        int total = 0;
        for (Integer num: nums) {
            total += num;
        }
        return total;
    }
}

```

Here, we can see in the code that we take values that are, in fact, integer objects and then perform addition operation on them. We know that they cannot be run on objects, but only on integer type primitive data values. The compiler does not return an error here and simply converts the objects to the required integer type primitive data. This code produces the following output:

Total is 36

This happens because the code is internally changed by the compiler. What happens is that the value stored in the object is read by the compiler using the `intValue()` method and is then used during the program. This is an excellent example of unboxing, because it finds the primitive value which is contained within the box of a Wrapper class object.

The language is therefore, often termed as a semi object-oriented language, because it is still capable of converting primitive data types and is not limited by only using objects for all functions. The relevant conversions occur in a smart manner, always providing the required functionality.

11.8 | Developing Logic

Developing logic is possible when we first understand the different programming tools that we can employ in Java. These tools include arithmetic operations, comparators, and the “if” based several types of statements. There are also loops which are great for developing simple logic and reducing the number of lines of code required to perform a specific task.

We will be discussing these elements and how their syntax is carried out in Java. We will start with the basic operators then move towards comparisons and loops, including nested looping programs.

11.8.1 Various Types of Operators



In the earlier Section 11.1.2, we have seen variables like `int`, `float`, `double`, `char`, `String`, etc. Variables are placeholders of specific types of data. For example, `int` holds numbers and `String` holds characters. For various numeric values such as `int`, `float`, and `double`, you are going to need to add them together, increment them, compare one to another, multiply them, divide them, etc. Java provides operators for these types of operations. An operator takes two values to generate result. These values are called *operands*. The operator stays in the middle of two operands. For example, the “+” operator adds two numeric values 4 and 6 to produce 10. So, the placement would look like “4 + 6”. “4” is the left operand, followed by “+” operator and then “6” operand. Figure 11.2 shows various operators supported by Java.

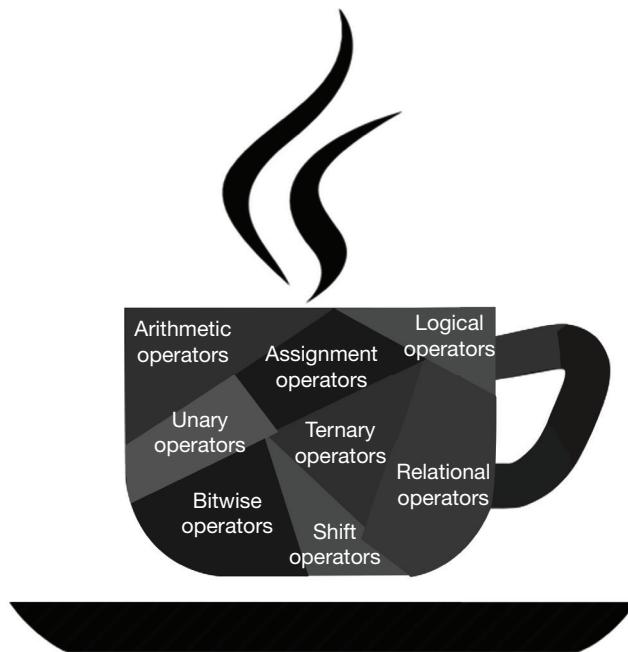


Figure 11.2 Operators in Java.

In C++, the operators are overloaded but this is not the case in Java. However, there are a few exceptions to this, where a few operators come overloaded in Java. For example, the “+” operator can be used to add two numeric primitives, or concatenate two operands if they are Strings. Similarly, the `|`, `^`, and `&` operators can be used in two different ways.

11.8.1.1 Assignment Operators

As the name suggest, assignment operator is used to assign a value to a variable. It may look like `int a = 10`. It may be simple but there will be situations which will demand complex expressions and casting. These involve primitive and reference variable assignments. Now let us understand what is variable. Variable is nothing but a bit holder with a specific type. For example, int holder, double holder, Boolean holder, and String holder. Each holder contains bits that represent value. Like for primitives, the bits represent numeric value. Similarly, objects also hold certain value. We will learn about object assignments in Section 11.7.1.2. Now, let us learn about primitive assignments.

11.8.1.1.1 Primitive Assignments

Similar to Math, Java uses “=” as an assignment operator to assign a value to a variable. There are various other types of assignment operators used in Java. A primitive variable can be assigned by a literal or an expression. For example,

1. `int a = 10; //literal assignment`
2. `int b = a * 10; // assignment with an expression including a literal`
3. `int c = a + b; // assignment with an expression`

In the first case of `int a = 10`, a literal integer 10 is implicitly an int. Now, let us explore some tricky cases. We all know that int is a 32-bit value and long is 64-bit. Hence, just as a bigger bucket can hold a smaller bucket, similarly long can hold an int value. Now, we all know that byte is an 8-bit value which is definitely smaller than int. However, what will happen if you try to change the type of a variable? For example,

```
byte a = 10;
```

Will this give us a compiler error? No. This will not give a compiler error as the literal value will be automatically cast by the compiler. The compiler will see the above code as follows,

```
byte a = (byte) 10; // Explicitly cast the int literal to a byte
```

This is also true for char and short type.

11.8.1.1.2 Primitive Casting

As we have seen in the above example, casting converts primitive values from one type to another. Casting can be implicit or explicit as we have seen earlier. Implicit cast means the compiler takes care of the conversion where you do not have to write code for the cast. Whereas in the explicit casting, you have to specifically write code for the cast. Implicit casting occurs only when you try to put smaller values into bigger ones like int into long, byte into int, etc. If you do not mention the casting then in the other way conversion compiler will through “possible loss of precision” error. This conversion like big value into smaller container for example long into int, int into byte etc. This type of conversion is called *narrowing*, which requires explicit cast. Let us look at some implicit and explicit casts.

Examples of implicit cast:

1. `int a = 10;`
2. `long b = a; // Implicit cast, an int value always fits in a long`

Examples of explicit cast:

1. `float f = 134242.003f;`
2. `int c = (int) f; // Explicit cast. In this case, we will lose the precision`

The following example is going to give a compiler error as it is expecting an explicit casting,

```
package java11.fundamentals.chapter11;
public class CastingError {
    public static void main(String args[]) {
        int a = 10.000; //going to through error
    }
}
```

The above program gives the following result.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  Type mismatch: cannot convert from double to int

at java11.fundamentals.chapter1.CastingError.main(CastingError.java:5)
```

To solve the above problem, you can cast the floating-point value explicitly to int but you are going to lose the precision.

```
package java11.fundamentals.chapter11;
public class CastingErrorSolution {
    public static void main(String args[]) {
        int a = (int) 10.000; //this will work
        System.out.println("Value of a : " + a);
    }
}
```

The above program produces the following result.

```
Value of a : 10
```

As you can see, the above casting results in loss of digits after the decimal. Now, let us see a few interesting cases. What if you cast a larger type to the smaller one? For example, casting long to byte. Let us study this with the following program and its output.

```
package java11.fundamentals.chapter11;
public class LargeToSmallCast {
    public static void main(String [] args) {
        long l = 100L;
        byte b = (byte)l;
        System.out.println("The value of byte b is " + b);
    }
}
```

The above code produces the following result.

```
The value of byte b is 100
```

Now let us increase the value of long variable, which is larger than the container size of byte. Let us see what will happen if we set value of long as 200.

```
package java11.fundamentals.chapter11;
public class LargeToSmallCastOuterRange {
    public static void main(String [] args) {
        long l = 200L; //this will give us unexpected result
        byte b = (byte)l;
        System.out.println("The value of byte b is " + b);
    }
}
```

The result shows that the code is giving us an unexpected result.

The value of byte b is -56

Why do we get a negative value? This happened because the value of long, which is narrowed, is too large for byte to hold. In this case, bits to the left of lower 8 go away. And after the removal of the lower 8 bits, if the value of the leftmost bit, which is the sign bit, turns out to be 1 then the primitive gives a negative value.

11.8.1.2 Object Assignments

We now understand if we say `int a = 10`, it means “*a* as int type holds value of 10 but in bits form”. But in the object world it is a little different. It is not the direct assignment of the value but a reference. For example, object *b* in the following example does not contain Book object itself but a reference to the object that sits on the heap. Hence, it is called *reference variable*.

```
Book b = new Book();
```

The above code does the following three things:

1. It creates a reference variable *b* of type Book.
2. It then creates a new Book object on the heap.
3. It assigns the newly created Book object to the reference variable *b*.

You may also assign a null value to the object by specifying it in the following way.

```
Book b = null;
```

In this case, the variable *b* holds bits representing “null”. This means that it creates space for the Book reference variable but does not actually create a Book object. This can be done by not explicitly referring to “null” as well. For example, `Book b;`

Please note that in method scope you must assign a value to a variable whereas in class scope if value is not assigned then variables get their default value. For example, `Book b;` can only be specified on the class level and not in any method. Variables inside methods called as local variables and do not carry any default values.

In the object-oriented world, we have a concept of superclass and subclass. Although you will be learning a lot about OOP in the Chapter 12, for this object reference concept, it is important for you to know a little bit about superclass and subclass. In Java, every class must originate from some class, which is called *superclass*. For example, a class called “Animal” can act like a superclass and holds standard animal properties, from which all other animals will be derived. Classes “Dog” and “Cat” can be derived from the superclass “Animal” so they will inherit all the properties of class “Animal”. Now that you understand the concept of superclass and subclass, we should try to understand the pool of object references. In this

world, a superclass object can refer to any object of its subclasses. For example, the Animal object reference can be used to refer to Dog object.

```
package java11.fundamentals.chapter11;
public class Animal {
    public void makeSound() {
        System.out.println("Animal makeSound method");
    }
}
```

The above code declares an Animal class. This class contains a method makeSound().

```
package java11.fundamentals.chapter11;
public class Dog extends Animal{
    public void makeSound() {
        System.out.println("Dog makeSound method");
    }
}
```

The above code declares a Dog class which extends Animal class. In this case, Dog is a subclass of the Animal class. In other words, Animal is the superclass of Dog.

```
package java11.fundamentals.chapter11;
public class ObjectReferenceExample {
    public static void main (String [] args) {
        Animal dogObj = new Dog(); // Legal declaration as Dog is a subclass of Animal
        Dog animalObj = new Animal(); // Illegal declaration as Animal is not a subclass
        of Dog
    }
}
```

The above program gives the following result.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from Animal to Dog
```

```
at java11.fundamentals.chapter1.ObjectReferenceExample.main(ObjectReferenceExample.java:6)
```

As you can see in the output, the error says, “Type mismatch: cannot convert from Animal to Dog”. It means you cannot assign a superclass object to a subclass variable. You can only assign a subclass object to a superclass variable. This is because a subclass Dog object is guaranteed to be able to do anything that a superclass Animal object can do. In other words, due to this superclass reference used for a Dog object, any variable within a Dog object can invoke Animal methods as this object holds the Animal reference. For example, a Dog object can invoke the makeSound() from the Animal class. However, in this case, we have overridden makeSound() method in the Dog class so with a Dog object, even with the Animal reference, the inherited method from Dog class will get called. See the following example,

```
package java11.fundamentals.chapter11;
public class ObjectReferenceMethodCallExample {
    public static void main (String [] args) {
        Animal dogObj = new Dog(); // Legal declaration as Dog is a subclass of Animal
        dogObj.makeSound();
    }
}
```

The above program gives the following result which shows that the Dog object's `makeSound()` method has been called.

Dog makeSound method

If we don't override `makeSound()` method in the Dog class, upon invoking `makeSound()` on the Dog object will invoke the Animal class's `makeSound()` method. See the following example:

```
package java11.fundamentals.chapter11;
public class DogWithoutOverridden extends Animal{}
```

In the above program, we have not added `makeSound()` implementation and hence, it will always use the Animal class `makeSound()` implementation.

```
package java11.fundamentals.chapter11;
public class ObjectReferenceMethodCallExample2 {
    public static void main (String [] args) {
        Animal dogObj = new DogWithoutOverridden(); // Legal declaration as Dog is a
        subclass of Animal
        dogObj.makeSound();
    }
}
```

The above program produced the following output which shows that the Animal class's `makeSound()` method has been invoked.

Animal makeSound method

However, think for a second, if the actual object is Animal and reference object is Dog, it would not know which `makeSound()` method to call. Hence this reverse assignment like subclass reference and superclass object does not work.

11.8.1.3 Arithmetic Operators

Arithmetic operators are mainly used to perform mathematical related operations. They are useful in performing operations like addition, subtraction, multiplication, division, and remainder. These operations are permitted on numeric data types like byte, short, int, long, float, and double. Table 11.7 shows the Arithmetic operators and their meanings.

Table 11.7 Arithmetic operators and their meanings

Operator	Meaning
+	Addition operator – Use for adding two numbers
-	Subtraction operator – Use for subtracting two numbers
*	Multiplication operator – Use for multiplying two numbers
/	Division operator – Use for dividing two numbers
%	Remainder operator – Use for getting remainder of two numbers

The following example demonstrates the use of these operators.

```
package java11.fundamentals.chapter11;

public class ArithmeticOperators {
    public static void main(String args[]){
        int x = 20;
        int y = 15;

        ArithmeticOperators ao = new ArithmeticOperators();
        int additionResult = ao.doAddition(x,y);
        int subtractionResult = ao.doSubtraction(x, y);
        int divisionResult = ao.doDivision(x, y);
        int multiplicationResult = ao.doMultiplication(x, y);
        int remainderResult = ao.doRemainder(x, y);
        System.out.println("Addition Result = " + additionResult + "\nSubtraction Result
= " + subtractionResult + "\nDivision Result = " + divisionResult + "\nMultiplication
Result = " + multiplicationResult + "\nRemainder Result = " + remainderResult);
    }

    public Integer doAddition(int x, int y){
        return x + y;
    }

    public Integer doSubtraction(int x, int y){
        return x - y;
    }

    public Integer doDivision(int x, int y){
        return x / y;
    }

    public Integer doMultiplication(int x, int y){
        return x * y;
    }

    public Integer doRemainder(int x, int y){
        return x % y;
    }
}
```

11.8.1.4 Unary Operators

Unary operators perform various operations such as increment, decrement, inverting the value of a Boolean, and negating an expression. It is quite convenient and widely used in loops. Table 11.8 shows the unary operators Java provides.

Table 11.8 Unary operators and their meanings

Operator	Meaning
+	Unary plus – It shows number as positive, but it is not needed as numbers are already positive
-	Unary minus – It inverts the sign of an expression
++	Increment operator – It increments value by 1
--	Decrement operator – It decrements value by 1
!	Logical complement operator – It inverts the value of a Boolean

The following program shows the use of the Unary operators.

```
package java11.fundamentals.chapter11;
public class UnaryOperators {
    public static void main(String args[]) {
        int a = 1;
        boolean b = false;
        System.out.println("Result of +a" + +a);
        // result is now -1
        System.out.println("Result of -a" + -a);
        // result is now 0
        System.out.println("Result of a--" + a--);
        // result is now 1
        System.out.println("Result of a++" + a++);
        // false
        System.out.println("Result of boolean b " + b);
        // true
        System.out.println("Result of complement boolean b " + !b);
    }
}
```

The above program produces the following result.

Result of +a1
Result of -a-1
Result of a-1
Result of a++0
Result of boolean b false
Result of complement boolean b true

11.8.1.5 Equality and Relational Operators

In many situations, as a programmer you need to check the equality of two operands to see if one is greater than, less than, equal to, or not equal to another operand. For this use case Java provides various operators, which are called equality and relational operators. These operators give result in Boolean, which is either true or false, and they are mainly used in decision-making and loops. Table 11.9 shows the equality and relational operators.

Table 11.9 Equality and relational operators

Operator	Description	Example
<code>==</code>	equal to	<code>10 == 20</code> is evaluated to false
<code>!=</code>	not equal to	<code>10 != 20</code> is evaluated to true
<code>></code>	greater than	<code>10 > 20</code> is evaluated to false
<code><</code>	less than	<code>10 < 20</code> is evaluated to true
<code>>=</code>	greater than or equal to	<code>10 >= 20</code> is evaluated to false
<code><=</code>	less than or equal to	<code>10 <= 20</code> is evaluated to true

The example below shows the use of these operators.

```
package java11.fundamentals.chapter11;
public class EqualityRelationalOperators {
    public static void main(String args[]) {
        int a = 10;
        int b = 20;
        if (a == b) {
            System.out.println("a == b");
        }
        if (a != b) {
            System.out.println("a != b");
        }
        if (a > b) {
            System.out.println("a > b");
        }
        if (a < b) {
            System.out.println("a < b");
        }
        if (a <= b) {
            System.out.println("a <= b");
        }
    }
}
```

The above program produces the following result.

<code>a != b</code>
<code>a < b</code>
<code>a <= b</code>

11.8.1.6 The instanceof Operator

The instanceof operator is useful in situations where you want to check if an object was instance of the specified class. As we have seen earlier, one of the OOP properties is inheritance, in which the subclass can be referred by a superclass reference. In such a scenario, it is sometimes important to check if the object belongs to the class you are expecting. In this case, the instanceof operator becomes useful, and it tells you if the comparing object is type of the class you are expecting.

The following example demonstrates the use of the instanceof operator. We will use the example classes “Animal” and “Dog” that we used earlier to demonstrate superclass–subclass relationship.

```

package java11.fundamentals.chapter11;

public class InstanceofOperator {
    public static void main(String args[]){
        Animal a = new Animal();
        Dog d = new Dog();
        if(a instanceof Animal){
            System.out.println("a is a type of Animal");
        }else{
            System.out.println("a is NOT a type of Animal");
        }
        if(a instanceof Dog){
            System.out.println("a is a type of Dog");
        }else{
            System.out.println("a is NOT a type of Dog");
        }
        if(d instanceof Animal){
            System.out.println("d is a type of Animal");
        }else{
            System.out.println("d is NOT a type of Animal");
        }
        if(d instanceof Dog){
            System.out.println("d is a type of Dog");
        }else{
            System.out.println("d is NOT a type of Dog");
        }
    }
}

```

The above program gives the following result.

```

a is a type of Animal
a is NOT a type of Dog
d is a type of Animal
d is a type of Dog

```

11.8.1.7 Logical Operators

Often times we come to a point in our program where we need to match various conditions to make a decision. For example, you may want to return a value if it falls between 10 and 20. In this case, you need an operator that will compare your value with two other values. This can be done with the help of logical operators. Table 11.10 shows all the logical operators and their descriptions. Logical operators return Boolean result, which helps compute the comparison. There are six logical operators: $\&$ (AND), $|$ (OR), \wedge (Xor), $!$ (Not), $\|\$ (conditional-OR), and $\&\&$ (conditional-AND). In the example below, we compare the value, say x , with 10 and 20.

```

if(x >= 10 && x <= 20) {
    return x;
}

```

These types of expressions called *compound expressions* as they let us combine two or more conditions into a single expression.

Table 11.10 Logical operators

Operator	Description
	conditional-OR – It returns true if either of the Boolean expression is true
&&	conditional-AND – It returns true if all Boolean expressions are true
!	Not – It returns false if true is passed and true if false is passed
	OR – It works similar to conditional-OR, where it returns true if at least one of the operands evaluates to true. But the difference is it evaluates both the operands before checking the OR condition.
&	AND – It works similar to conditional-AND, where it returns true if both of the operands evaluate to true. But the difference is it evaluates both the operands before checking the AND condition
^	Xor – It returns true if only one of the operands evaluates to true

Notes:

1. False || true expression is evaluated to true.
2. False && true expression is evaluated to false.

The following example shows the uses of logical operators.

```
package java11.fundamentals.chapter11;
public class LogicalOperators {
    private int a = 10;
    private int b = 20;
    public static void main(String args[]){
        LogicalOperators lo = new LogicalOperators();
        lo.checkBetweenAB(15);
        lo.checkBetweenAB(21);
        lo.checkBetweenAB(9);
        lo.checkLessAOrLessB(15);
        lo.checkLessAOrLessB(8);
        lo.checkLessAOrLessB(25);
        lo.checkLogicalNot(true);
        lo.checkLogicalNot(false);
    }
    public void checkBetweenAB(int x){
        if(x >= a && x<=b){
            System.out.println("The number " + x + " is between " + a + " and " + b);
        }else{
            System.out.println("The number " + x + " is outside the range of " + a + " and "
+ b);
        }
    }
    public void checkLessAOrLessB(int x){
        if(x < a || x < b){
            System.out.println("The number " + x + " is less than " + a + " or " + b);
        }else{
            System.out.println("The number " + x + " is greater than " + a + " and " + b);
        }
    }
    public void checkLogicalNot(Boolean y){
        if(y){
            System.out.println("y is true");
        }
        if(!y){
            System.out.println("y is false");
        }
    }
}
```

The above program produces the following result.

```
The number 15 is between 10 and 20
The number 21 is outside the range of 10 and 20
The number 9 is outside the range of 10 and 20
The number 15 is less than 10 or 20
The number 8 is less than 10 or 20
The number 25 is greater than 10 and 20
y is true
y is false
```

As you can see, the logical AND operator makes sure that both the conditions are true whereas the logical OR operator checks if either of the conditions is true. In the OR check scenario, if the first condition satisfies then it will not check for the second condition and simply return the result. In the AND check scenario, it will check for the second condition only if the first condition satisfies. If the first condition does not satisfy then it will break the “if” block.

11.8.1.8 Ternary Operator

The ternary operator allows you to shorten the if-then-else statement. Although the code is less readable than the actual if-then-else block, it helps to write code faster. The syntax is,

```
variable = Expression ? value1 : value2;
```

This is a handy operator and it is very easy to understand. If the Expression is true, value1 is assigned to variable and if the Expression is false, value2 is assigned to variable. Let us see an example,

```
package java11.fundamentals.chapter11;
public class TernaryOperator {
    public static void main(String args[]) {
        int a = 10;
        int b = 20;
        boolean c;

        //Following expression checks if the value of a is less than b
        c = a < b ? true : false;

        System.out.println("Value of c is " + c );
    }
}
```

The above program produces the following result.

```
Value of c is true
```

11.8.1.9 Bitwise and Bit Shift Operators

Bitwise and bit shift operators are a bit uncommon for use in programs. However, they are very useful in some situations. As the name suggests, these operators perform bitwise and bit shift operations on integral types. Table 11.11 lists the bitwise and bit shift operators.

Table 11.11 Bitwise and Bit shift operators

Operator	Description
<code>~</code>	Bitwise complement – It inverts the bit pattern, such as making every “0” to “1” and every “1” to “0”
<code><<</code>	Left shift – It shifts a bit pattern to left
<code>>></code>	Right shift – It shifts a bit pattern to right
<code>>>></code>	Unsigned right shift – It shifts a zero into the leftmost position
<code>&</code>	Bitwise AND – It performs bitwise AND operation
<code>^</code>	Bitwise exclusive OR – It performs bitwise exclusive OR operation
<code> </code>	Bitwise inclusive OR – It performs bitwise inclusive OR operation

The following example demonstrates the use of bitwise and bit shift operators.

```
package java11.fundamentals.chapter11;
public class BitwiseBitshiftOperators {
    public static void main(String[] args) {
        int x = 30;
        int y = 60;
        int z = 300;
        int k = 55;
        int l = 15;
        int output;
        output = ~k; //performs bitwise complement operation. It simply inverts the bit
        pattern by changing every 0 to 1, and every 1 to 0.
        System.out.println(output);
        output = x ^ y; //performs bitwise Xor operation where it compares the close by
        bits. If found same, it returns 0 or else 1.
        System.out.println(output);
        output = x & y; //performs bitwise AND operation where it compares the close by
        bits. If found both bits as 1, it returns 1. If either of the bit is not 1 then 0.
        System.out.println(output);
        output = x | y; //performs bitwise OR operation where it compares the close by
        bits. If found either of the bit 1, it returns 1 or else 0.
        System.out.println(output);
        //perform signed left shift operation. It shifts a bit pattern to the left by
        number of specified bits. The low-order positions bits are filled with zero bits.
        System.out.println(z << 0);
        System.out.println(z << 2);
        System.out.println(z << 6);
        //perform signed right shift operation. It shifts a bit pattern to the right by
        number of specified bits.
        System.out.println(z >> 0);
        System.out.println(z >> 1);
        System.out.println(z >> 6);
        System.out.println(l >>> 1); //perform unsigned right shift operation
    }
}
```

**QUICK
CHALLENGE**

Create a chart to show all the operators with examples.

11.8.2 Expressions

Expressions are a collection of methods, variables, and operators that are constructed according to Java's specific syntax to evaluate values and provide answers. In all the examples in this chapter, we have often used these expressions, which we are defining here in terms of their role in creating logical functionality in a Java program. Here, we present a few expressions that use different operators and methods to cover all types of instances.

```

1. package java11.fundamentals.chapter11;
2. public class ExpressionsExample {
3.     public static void main(String args[]){
4.         int num = 0;
5.         int num1 = 1;
6.         int num2 = 2;
7.         int val1 = 4;
8.         int val2 = 4;
9.         int arrayOne[] = new int[10];
10.        arrayOne[1] = 50;
11.        System.out.println("This array position of index 1 presents value of: " +
arrayOne[1]);
12.        int res = num1 + num2;
13.        if (val2==val1){
14.            System.out.println("All values are equal");
15.        }
16.    }
17. }
```

These are a few instances which describe how to set up expressions. Setting expressions are important because they often control the statements that come after them, depending on the logic which was generated and processed by the expression. Line 4 code shows the correct initialization of a variable `num` which is then equated to an integer of 0. On line 10, a value is saved in the second indexed location of an array, which is then printed on the console in the next line of code.

Line 12 presents a mathematical operation where two numbers are added and their sum is stored in another variable of type integer. Since the result is of integer type, we may find that in an actual program, there is a need for type promotion if `num1` and `num2` are not of matching datatypes. On line 13, we perform a logical comparison of two variables and then only on line 14 print a message to the console, if both variables have equal values. Following image shows the result of the above program.

This array position of index 1 presents value of: 50
All values are equal

Expressions are important as they provide a controlling logic. The order of operations can often be ambiguous if we do not follow ideal mathematical practices. Therefore, it is important to understand the best ways to present mathematical operations, just as we would present them traditionally. Take the example of using brackets to describe clear information, as given in the examples below.

`(x+y) /n;`

In the above example, as per the operator precedence, Java will first compute the addition and then division by `n`.

`(x/n) +y;`

In the above example, as per the operator precedence, Java will first compute the division and then addition with `y`.

`x+ (y/n) ;`

In the above example, as per the operator precedence, Java will first compute the division of y by n and then addition with x .

The use of simple brackets in the above examples provide an ideal control over the results of the mathematical operations. There can be a number of brackets and order of operations that always ensure that it is not possible to have a wrong result, which can destroy the program logic altogether.

11.9 | Control Flow



Control flow is an important element in any programming language. The Java syntax also allows programmers to use statements which ensure that the program direction and execution can be controlled based on the choices available during program execution. Figure 11.3 shows all the control statements. We will discuss all these different ways in which the control statements can be used. Below we will see how “if” can provide control, similarly how “switch” can be used to control the flow which we have already seen in Section 11.5.

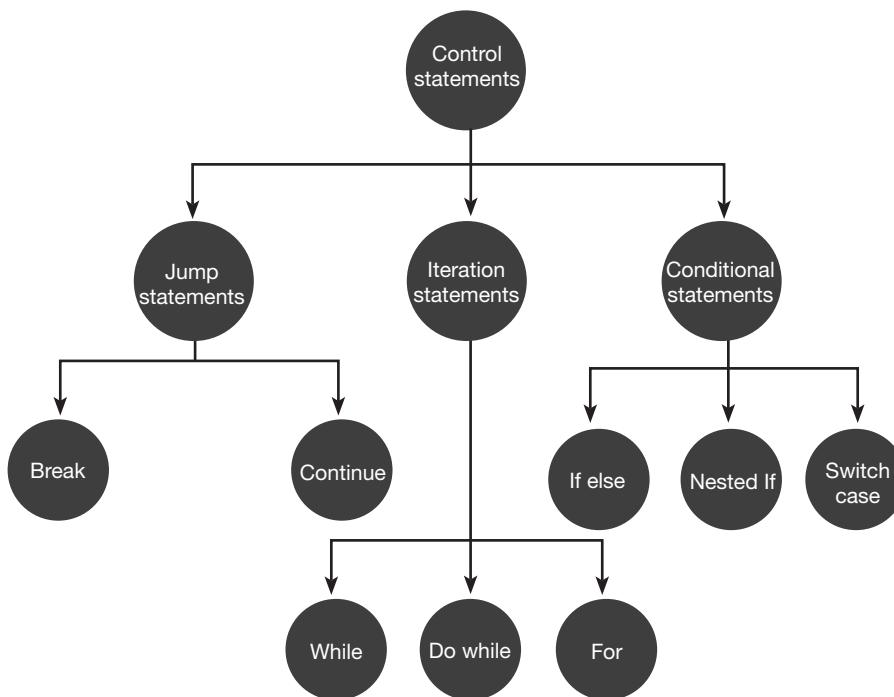


Figure 11.3 Control statements.

The “if” statement can be termed as a legacy statement, as it is present universally in almost all programming languages. It provides the basic control flow by ensuring that a particular program function is only performed if the tested condition, which is predefined in the statement, is found to be true by the compiler. Below is a simple example which shows its individual use.

```

package java11.fundamentals.chapter11;
public class AgeCompare {
    public static void main(String[] args) {
        int age = 10;
        System.out.println("The right age for Middle School is: " + age);
        if (age < 14) {
            System.out.println("The current age is less than high school age");
        }
    }
}
  
```

The above program is excellent in providing the control flow use. The system sets up a variable fixed at the value of 10. When the if statement runs and finds that the age is less than 14, it executes the next code block. It is also possible to represent a single statement

without using the brackets. However, the ideal Java syntax practice is to use the brackets to create code blocks. In most situations, you will need to carry out a set of functions if the condition is found to be true. Following is the result of the above program.

The right age for Middle School is: 10
The current age is less than high school age

However, we find that the if statement remains limited if the condition is found to be false. A false condition will mean that the next statement or the set of code in the brackets will be ignored. There are times where it is important to carry out specific executions according to the results of the Boolean expression placed in the parenthesis of the if statement. For such scenarios, we use the “if-else” combination. Below is a good syntax example for this control flow statement.

```
package java11.fundamentals.chapter11;
public class AgeCompare2 {
    public static void main(String[] args) {
        int age = 15;
        System.out.println("Jane is " + age + " years old");
        if (age < 14) {
            System.out.println("She's is too small to be in high school");
        } else {
            System.out.println("Jane needs to be in high school according to her age");
        }
    }
}
```

The above code will produce the following result.

Jane is 15 years old
Jane needs to be in high school according to here age

This example shows that the failure of the if expression will execute the code block present with “else”. It is also possible to set up a long chain of “if-else-if” conditional statements. These statements can provide a series of actions, where they successfully follow a method of checking all possible conditions and produce the intended results.

It is better to stay away from this kind of conditional Java syntax as there is a great chance of producing logical errors that are hard to find even with the best IDE software tools for Java programming. We can use the “switch” statement when there can be multiple values and situations that need to be attended. Below is a program that shows how to use this method:

```
package java11.fundamentals.chapter11;
public class SwitchExample {
    public static void main(String[] args) {
        int numSides = 5;
        switch (numSides) {
            case 3:
                System.out.println("It is a triangle");
                break;
            case 4:
                System.out.println("It is a quadrilateral");
                break;
            case 5:
                System.out.println("It is a pentagon");
                break;
            case 6:
                System.out.println("It is a hexagon");
                break;
            default:
                System.out.println("It is a polygon with more than six sides");
        }
    }
}
```

The above program will produce the following result.

It is a pentagon

This program checks the number of sides of a shape to present the intended method. Clearly, there can be further cases that can easily be added in this switch code block. An advantage in this program is that the order can be arbitrary and therefore, it is easier to avoid mistakes or to find out if a particular condition is catered by the program.

11.10 | Loops

Loops are closely associated in programming languages with conditional operators. They control the program flow. Figure 11.4 shows the loops execution steps. We will discuss two types of loop statements that you can use in Java syntax – while loop and for loop.

11.10.1 While Loop

The “while” loop is by design an infinite loop. The expression presented in its parenthesis provides a Boolean answer true or false. This expression evaluation of Boolean value is perfect for ensuring that the loop can also work as the conditional operator.

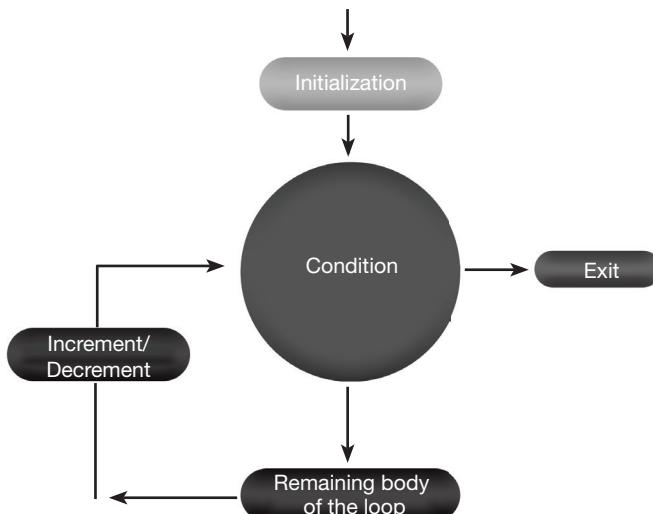


Figure 11.4 Loops execution steps.

There are two versions of the “while” loop. The first version is simple, where the expression is first tested and then the code block following the loop command is executed. The other version is the “do while” loop, in which it is possible to first run the code block and then test out the expression present in the while loop statement. Below is an example that describes loop sequence.

```

package java11.fundamentals.chapter11;
public class WhileExample {
    public static void main(String[] args) {
        int numCount = 1;
        while (numCount < 11) {
            System.out.println("Current Count is: " + numCount);
            numCount++;
        }
    }
}
  
```

The above program will print the value of numCount which initialized to 1 before entering into the while loop. The while loop has condition which checks if numCount is less than 11. If the condition is satisfied the program execution goes to the next line which prints the value of numCount. On the next line, the value of numCount increases by 1 and this way the loop continues until numCount is 10. Till this point, the program has already printed numbers from 1 to 10. After printing the numCount with value 10, the next line increases the numCount value by 1 which makes it 11. When the execution reaches to while, the expression check the condition and since the value of numCount is 11 which is not less than the comparison value 11, this makes the expression return false. Hence the execution exits the while loop.

```
Current Count is: 1
Current Count is: 2
Current Count is: 3
Current Count is: 4
Current Count is: 5
Current Count is: 6
Current Count is: 7
Current Count is: 8
Current Count is: 9
Current Count is: 10
```

Remember, it is always possible to set up an arbitrary condition in the loop, which will create an indefinite loop in the Java program. Using the “do” is a special case, which allows the program to at least execute the code block once, even if the “while” expression is found to be false in its first instance.

11.10.2 For Loop

The “for” loop statement is found in most legacy programming languages. This loop usually works with a counter and executes a block of code a finite number of times, which is specifically defined. It is an excellent method of reducing the overall code length, and ensures the possibility of several mathematical functions, which are required in several Java programs.

The “for” loop has a characteristic expression section in parenthesis. The first element initiates the counter variable, the second provides the loop termination condition, and the third element controls the number of iterations of the loop with an increment or a decrement. The code block that follows is the one which is executed in each loop iteration. Below is a simple example to help you learn its syntax.

```
package java11.fundamentals.chapter11;
public class LoopExample {
    public static void main(String[] args) {
        for (int count = 1; count < 6; count++) {
            System.out.println("The current loop iteration number is: " + count);
        }
    }
}
```

This program will run five times and present the iteration numbers from 1 to 5, each time carrying out a single increment. See the following output.

```
The current loop iteration number is: 1
The current loop iteration number is: 2
The current loop iteration number is: 3
The current loop iteration number is: 4
The current loop iteration number is: 5
```

However, Java also offers an enhanced way of running the count in the loop, which eliminates common problems that appear with the use of loops. Below is an example of this type of “for” loop.

```
package java11.fundamentals.chapter11;
public class BetterLoopExample {
    public static void main(String[] args) {
        int[] count = { 1, 2, 3, 4, 5 };
        for (int num : count) {
            System.out.println("The Count now is: " + num);
        }
    }
}
```

This program will also print the line five times, see the following output.

```
The Count now is: 1
The Count now is: 2
The Count now is: 3
The Count now is: 4
The Count now is: 5
```

This type of loop works by running the number of counts as many times as there are values in the employed integer array. This practice ensures that the control can have an independent layer, allowing programmers to stay away from simple mathematical and logical errors.



Can you have a nested loop? (Hint: Nested loops are ones that contain another loop inside them.)

11.11 | Branching

It is possible to have improved control by using branching statements. Java offers two options for this function. This includes the use of “break” and “continue” statements. We have already presented an unlabeled form of break, which we can use to terminate any loop, as well as the different cases of switch, where this statement ends the execution of switch code block.

Let us take a look at a detailed example that shows the use of branching with other control flow elements.

```
package java11.fundamentals.chapter11;
public class DemoForBreak {
    public static void main(String[] args) {
        int[] intArray =
            { 68, 25, 74, 33, 95, 17, 53, 28, 1986, 53 };
        int reqNum = 17;
        int count;
        boolean found = false;
        for (count = 0; count < intArray.length; count++) {
            if (intArray[count] == reqNum) {
                found = true;
                break;
            }
        }
        if (found) {
            System.out.println("We found " + reqNum + " at the array index of " + count);
        } else {
            System.out.println(reqNum + " is not stored in the array");
        }
    }
}
```

This program is an excellent example of Java syntax structures which produces the following output.

We found 17 at the array index of 5

The program initially describes an array with multiple values. We then create a counter that runs all these values one by one, to find if any value exactly matches the required number. The for loop is broken right then by the break command, showing an excellent use of the required branching. The ideal message can then be displayed by using another conditional statement. The program will iterate only if the value is continuously not found in the branched if statement.

Another way that “break” is employed is in a labeled form. In this form, it is designed to eliminate a specific case, which may be present as an external element in the program execution. Below is a program that explains this specific use of branching.

```
package java11.fundamentals.chapter11;
public class LabelBreakDemo {
    public static void main(String[] args) {
        int[][] arrayOne = {
            { 24, 87, 3, 18 },
            { 12, 76, 2000, 19 },
            { 22, 112, 67, 655 }
        };
        int lookFor = 12;
        int i;
        int j = 0;
        boolean numFound = false;
        search:
        for (i = 0; i < arrayOne.length; i++) {
            for (j = 0; j < arrayOne[i].length; j++) {
                if (arrayOne[i][j] == lookFor) {
                    numFound = true;
                    break search;
                }
            }
        }
        if (numFound) {
            System.out.println("We Found " + lookFor + " at " + i + ", " + j);
        } else {
            System.out.println(lookFor + " not in this array");
        }
    }
}
```

According to the way that we have included 12 in this array, we can clearly see that it is in the second set of curly braces for the integer values. This describes that *i* will have the index of 1 in this position, while *j* will have the 0 position as 12 is the first element at the array position. This means that this program will print the following result.

We Found 12 at 1, 0

This clearly describes that the loop did not run any longer when the loop value identified 12, as the break statement eliminated the entire “search” code block. To get more control, it is possible to employ another branching statement of “continue”. This is excellent for branching loops and calculating particular elements. This is evident in the following example.

```

package java11.fundamentals.chapter11;
public class DemoForContinue {
    public static void main(String[] args) {
        String searchP = "peter piper picked a peck of pickled peppers";
        int total = searchP.length();
        int numP = 0;
        for (int i = 0; i < total; i++) {
            // only want to count p's
            if (searchP.charAt(i) != 'p')
                continue;
            numP++;
        }
        System.out.println("We found a total of " + numP + " p's.");
    }
}

```

This program only counts character *p* when it is present, and simply moves back to increment *p* if the character is not found.

We found a total of 9 p's.

There is another method called as “return”, which is used to stop a method and return the execution to the code, which follows the method invocation. See the following example which shows the use of “return”. As soon as “return” is encountered the program execution exits the method.

```

package java11.fundamentals.chapter11;
public class DemoReturn {
    public static void main(String[] args) {
        String searchP = "peter piper picked a peck of pickled peppers";
        int total = searchP.length();
        int numP = 0;
        for (int i = 0; i < total; i++) {
            // only want to count p's
            if (searchP.charAt(i) != 'p')
                continue;
            numP++;
        }
        if(numP > 5) {
            System.out.println("We found more than 5 p's. Hence exiting the method.");
            return;
        }else {
            System.out.println("We found a total of " + numP + " p's.");
        }
    }
}

```

The above example is modified version of our earlier “continue” example. It contains a “if” condition which checks if numP is larger than 5 and if so it exits the method with the help of “return”. See the following output of the above program.

We found more than 5 p's. Hence exiting the method.

Summary

We have described all the important elements that make up the Java syntax. These structures identify the presence of coding elements, blocks, and other facilities that are employed when writing code in Java. We also discussed specific options of Java such as the presence of static and non-static elements. The ability of the language to perform autoboxing and unboxing is also important. Java provides an excellent ability to develop program logic and carry out counting, arithmetic, and other related operations using a variety of code structures. We believe that this chapter will ensure that you have enough knowledge to become an efficient Java programmer. In this chapter, we have learned the following concepts:

1. Structure of a Java Program.
2. Identifiers, keywords, primitive classes, literals, variables, code blocks, and comments.
3. Static and non-static methods.
4. Enums and various Java operators.
5. Wrapper classes and how to use them.
6. Autoboxing and unboxing.
7. Expressions and loops and control flow.

In the Chapter 12, we will study the object-oriented programming world. We will cover abstraction, encapsulation, inheritance, and polymorphism in detail, accompanied by various examples, and overloading and overriding.

Multiple-Choice Questions

1. How many bits are in long?
 - (a) 8
 - (b) 12
 - (c) 64
 - (d) 20
2. How to start a block comment?
 - (a) /
 - (b) !-
 - (c) *
 - (d) /*
3. In Java, which of the following is not considered a numerical type?
 - (a) int
 - (b) char
4. The `System.in.read()` can be utilized for reading a character from keyboard.
 - (a) True
 - (b) False
5. In Java language, which one of the following is not an integer value?
 - (a) 12
 - (b) '23'
 - (c) 10
 - (d) 100

Review Questions

1. What is autoboxing and unboxing? Explain with examples.
2. What is the maximum number int data type can hold?
3. What is the use of enum?
4. What is literal? Give some examples.
5. Define a for each loop with an example.
6. What are the benefits of autoboxing and unboxing?
7. When should we use StringBuffer?
8. What is the use of instanceof operator?
9. Can you use Integer to hold long value?
10. Is StringBuilder immutable?
11. Can you pass int value to a method which accepts float?

Exercises

1. Write a program to add digits from 1 to 9 and repeat this process 10 times.
2. Write a program to create String array and display each element.
3. Write a program to use StringBuffer and StringBuilder. Print the time taken by each method and find out the fastest one. Explain why one is faster than the other.

Project Idea

Write a program to create a book library. Identify the classes you need to build. Create a String array to add book names and use this to display the required book information. Write

a method to add a new book to this array and similarly add, remove, and update methods. Write a method that will print all the books in the library (from the String array).

Recommended Readings

1. Kathy Sierra and Bert Bates. 2018. *OCA Java SE 8 Programmer I Exam Guide*. McGraw Hill Education: New York
2. R. Nageswara Rao. 2016. *Core Java: An Integrated Approach*. Dreamtech Press: New Delhi

Object-Oriented Programming

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- History of object-oriented programming.
- Principles of object-oriented programming.
- Encapsulation and how it works in practice.
- Inheritance and how it works in practice.
- Polymorphism and how it works in practice.
- Abstraction and how it works in practice.
- Abstract class and how to use it in your program.
- Interface and how it is useful in designing inheritance.
- Future of object-oriented programming.
- Overloading and its use.
- Overriding and its use.

12.1 | Introduction

Programming languages allow computers to offer varied functionality. They enable us to create specialized applications that are capable of providing advanced functionality. There are different programming schemes that are used by different programming language platforms. This chapter focuses on the principles of object-oriented programming, or OOP in short.

This chapter is divided into relevant sections. We will start by discussing basic OOP principles, which define the use of objects in programming languages such as Java. We will then discuss the use of classes and objects for implementing the same functionality, as produced by more direct languages such as C. We will discuss various subtopics that are essential for building a key understanding of the OOP world.

We will first discuss the basic principles of OOP languages and then study the other important themes that you must explore to learn more about the OOP world. Let us look at the history of programming principles, which shows how it was gradually possible to reach the OOP design of programming paradigm.

12.1.1 History of Object-Oriented Programming

The concepts of using objects and providing an orientation to computer code were developed in the early 1960s, especially by researchers at MIT. Programmers referred to code elements as objects, which have a set of properties. Simula is the first language that presented the concepts of classes and objects, which are important for defining a language as one that employs OOP principles.

OOP languages improved data security and allowed programmers to produce data encapsulation in the form of creating private and public variables. Such compilers became popular and were employed to create programs for mainframe computers. The early OOP tools were especially efficient when used for carrying out complex tasks.

OOP principles especially got in popular use as they were perfect for creating programs that could follow human language-form instructions. The concepts of inheritance and encapsulation really caught on with various programmer communities, which understood that there were several benefits of switching from function-based programming especially for complex tasks.

The early languages started with smart ways to describe objects and create situations where it is possible to hide the implementation of the code to other programmers and program users. The programming environments gradually became available to smaller working situations.

There are two approaches that became widely used, with other approaches being dropped. Functional programming and OOP became the paradigms that were employed in all popular languages. There are many differences between functional

programming and OOP. One of the major differences is that functional programming follows stateless programming model and OOP follows stateful programming model.

C++ is an object-oriented language that was prepared from C, which is a fundamentally functional programming language. Software construction still followed instances where functional languages formed the basis of development while allowing programmers to use tools and libraries that implemented objectivity in their programming use.

OOP concepts gradually became popular as it was important to improve code security and provide control over data interfaces and class implementations. The OOP concepts gave rise to design patterns that are now commonly employed to resolve software problems in the modern development environment.

The use of behaviors and inheritance are the primary factors to incorporate an object-oriented design. This is a situation where it is possible to employ polymorphism and ensure the use of mutable objects. There are various methods that are still under the basic foundation of object-oriented principles. This may include abstraction, prototyping, and the use of singleton structures.

OOP languages have already surpassed the use of functional languages in modern use. These days, they are competing with the use of relational databases, as this ensures that it is possible to resolve all issues. According to computer scholars, the OOP paradigm is perfect when developers must create software systems that resemble human elements. It is perfect for taking a natural approach towards resolving problems by creating objects, which achieve the objectives required to resolve each scenario.

QUICK CHALLENGE

Create a comparison chart to differentiate between procedural programming and object-oriented programming.



12.2 | Object-Oriented Programming Principles

OOP languages use the concept of creating every functionality with the use of distinct objects. This is different from the principle of creating functions that hold independent value. OOP principles call for creating data objects that provide a higher level of functionality and make it easier for the program developer to prepare a project according to the specific requirements of the client. Figure 12.1 shows the object-oriented principles.

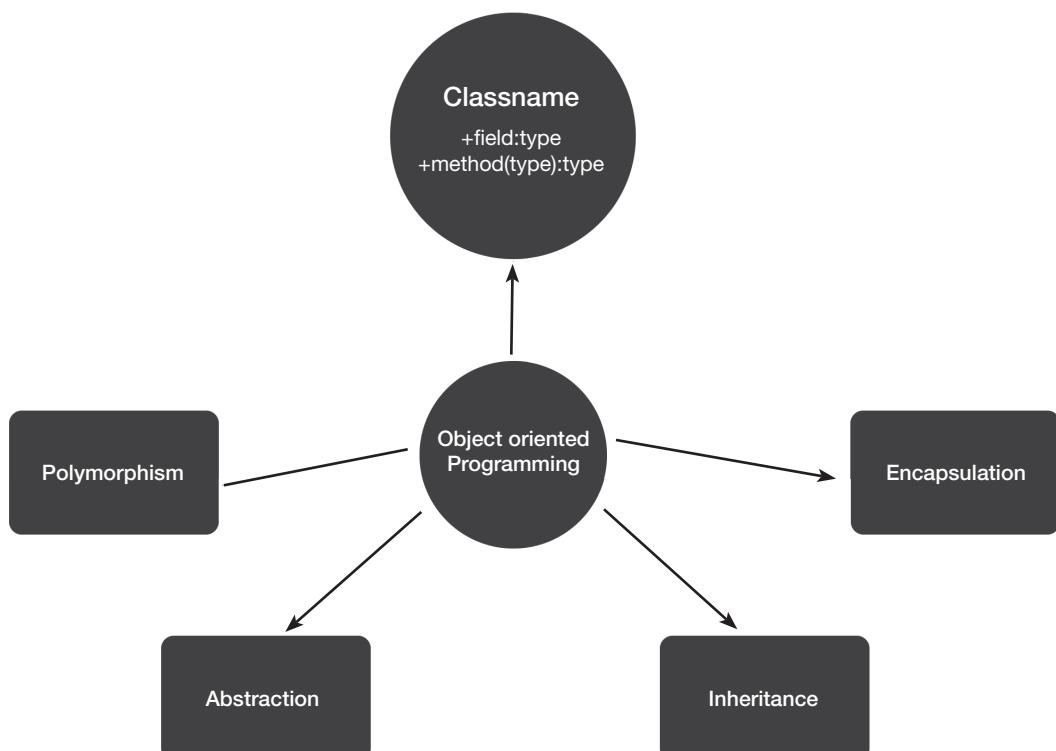


Figure 12.1 Object-oriented programming principles.

Regardless of any OOP language, there are four principles that define this technique.

- 1. Encapsulation:** This is a method that separates data implementation from the user.
- 2. Abstraction:** It describes the use of simple class objects that may provide the most complex of functions. This is a key principle of these languages and is often combined with encapsulation for an easier understanding and practical application.
- 3. Inheritance:** It allows the creation of data hierarchy structures, which ensures that data objects can form related trees and branches. It creates the system of classes that are built within other classes to follow a systematic relationship for defining instances, variables, and implementing the functionality from all the upper level classes. In other words, every class that has super class gets access to variables and methods from all the super classes at every level.
- 4. Polymorphism:** It is a more difficult concept to understand. In simple terms, it means that a single object may take on different forms, according to the defining principles of its use in the programming language.

Now, we will discuss these important concepts in greater detail. This will help you build a better understanding of the OOP concepts, and allow you to figure out how to go about using OOP languages for programming and application development.

12.2.1 Encapsulation

The first concept is the encapsulation of the data. Since data is arranged in the form of defined objects, they hold the properties of being distinct in their structure and is fully self-contained. The inner working of an object is defined by its state (attributes) and remains invisible to other parts of the code. The objects can show their behavior but maintain a strong boundary that separates them from other objects that are present in the programming environment.

Encapsulation works on the principle of hiding. The inner structure of all data objects is distinct. It contains all the elements, which are required to process it as a standalone part of the programming code. This objective is achieved by implementing boundaries that protect objects using specific tools. Access modifiers are used in a language such as Java, which allows you to hold full control of the attributes that define a data object. Figure 12.2 shows an example of encapsulation where all the ingredients are hidden inside the spring role. This can help you to visualize how encapsulation works.

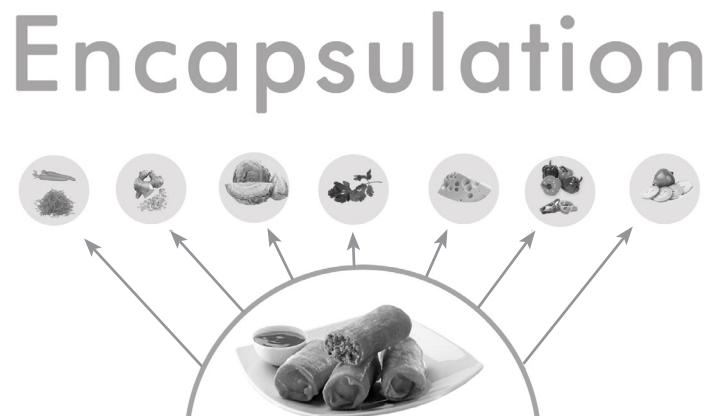


Figure 12.2 Encapsulation example.

There are some languages that create very strong encapsulation boundaries. Then there are languages like Java, which provide better control over the object structure by allowing programmers to set up different object property specifiers. However, encapsulation is still a primary feature in Java too.

This important principle allows us to separately keep data and the code sections that call for it. It allows programmers to change the original code, while never affecting the database objects that hold the legacy data, since they are always called but remain inaccessible for change due to a specific access structure.

Here is an example that shows how we implement the principle of encapsulation in Java by producing functionality, which remains locked with a boundary that separates the outside world from the class behaviors.

```

package java11.fundamentals.chapter12;
class DemoEncap {
    private int ssnValue;
    private int employeeAge;
    private String employeeName;
    // We will employ get and set methods to use the class objects
    public int getEmployeeSSN() {
        return ssnValue;
    }
    public String getEmployeeName() {
        return employeeName;
    }
    public int getEmployeeAge() {
        return employeeAge;
    }
    public void setEmployeeAge(int newValue) {
        employeeAge = newValue;
    }
    public void setEmployeeName(String newValue) {
        employeeName = newValue;
    }
    public void setEmployeeSSN(int newValue) {
        ssnValue = newValue;
    }
}
public class TestEncapsulation {
    public static void main(String args[]) {
        DemoEncap obj = new DemoEncap();
        obj.setEmployeeName("Mark");
        obj.setEmployeeAge(30);
        obj.setEmployeeSSN(12345);
        System.out.println("Employee Name is: " + obj.getEmployeeName());
        System.out.println("Employee SSN Code is: " + obj.getEmployeeSSN());
        System.out.println("Employee Age is: " + obj.getEmployeeAge());
    }
}

```

In this example, we have three private variables that are described during class initialization. They remain private and cannot be affected by the methods used in a typical program. However, it is possible to use set and get methods to define these variables in a particular class, where they can be mentioned and used in the main method of the program. This will produce instances that create new data objects and hold value in them, while the actual initialization and the coding behind the variables remain separated in the class definition.

This program will print *Mark*, *30*, and *12345* from the object getting values, without ever affecting the definition of the variables or making it available for the program to alter them.

Encapsulation is slightly different from abstraction in the manner that it defines the combination of all concepts into a single item with the ability to hide its internal data, which is not directly accessible to a program user. Encapsulation creates low coupling where various code elements do not have to depend entirely on each other. This is an excellent programming practice, which efficiently uses the available resources and is achieved through this ideal OOP principle.

Encapsulation is also excellent in terms of allowing the data and functionality to remain available for a user, while still hiding the way it is implemented. There is no information about the way objects are supposed to work, while still understanding the data that it demands, and the methods that it contains to get the job done.

**QUICK
CHALLENGE**

Based on the above-mentioned example of `DemoEncap`, write a program on a real-life problem to demonstrate encapsulation.

12.2.1.1 Advantages of Encapsulation

There are several advantages of encapsulation and they are the reasons for introducing languages that employ OOP principles. Here, we share the details of some important benefits:

1. It provides the advantage of creating flexible code. This is possible because we can implement a variable field in any way we want throughout the program. We can only use the different methods that remain within the classes that we have implemented. The class can be maintained directly, while various implementations occur as we see fit.
2. We can ensure that fields can be only either read or written. This is possible by avoiding the getter and setter methods. We can create a variable as a private one and then only use the get method. This will ensure that there will never be a change in the value of a particular field. Similarly, we will only use the set method to lock the variable value in only a written situation.
3. A program user will never know how the code works in the background for any variable present in an encapsulated class structure. This means that they only have access to employing get and set methods, where they want to bring value or set value to the variables. There is no way of finding out how the actual value would be assigned or read from the original variable, which remains private in a language like Java.



What are the disadvantages of encapsulation?

12.2.2 Abstraction

Abstraction is a concept which is best defined and described in line with encapsulation. It suggests that it is possible to develop classes and objects that are defined according to their functionality, rather than their programming implementation. This leaves us in terms of creating models for all our requirements.

These models serve as structures and never represent the presence of an actual item. The primary characteristics that define an object completely distinguish it from the other objects. Abstraction is also produced by creating conceptual boundaries for these defining characters, which ensures that all objects are understood by the viewers according to their specific perspective.

Abstraction means that we can create programming tools that we can employ multiple times. It defines an object without ever presenting the object for its instance. This can be understood by creating classes that effectively deal with related items such as recording the names and addresses of employees by using a class, which is designed for handling personal information.

Abstraction in Java can be best defined by using abstract classes. These are classes that you specify by adding the word “abstract” before mentioning the name of the class. This will lock the class and tell the compiler that it cannot be instantiated ahead. It is simply defining a class as an incomplete one. However, the incompleteness can be present in any part of the class definition. This is again identified with the keyword “abstract” with the method that we want to leave undefined. Here is a simple example:

```
abstract class MusicInstruments {
    protected String nameofInstrument;
    abstract public void playInst();
}
```

On its own, this will be an abstract object that has a defined structure but actually points to particular code or functionality in a program. However, we use this abstract by creating an extension that produces a subclass:

```
abstract class NewInstrument extends MusicInstruments {
    protected int numberOfPieces;
}
```

Now, this is a creation of a subclass that defines some object of value which has an additional field, aside from the initial variable fields. We can now create a set of subclasses that can all add various values and can be used for an actual implementation in a

Java program. Java also has the capability of producing interfaces, which allow for the use of the same abstraction principle to carry out the intended functionality that remains hidden from the common users.



Can a class extend multiple abstract classes?

12.2.2.1 Use of Abstract Classes

As we have seen above, Abstract classes set up abstract methods, which cannot have an implementation. You can have an abstract class which does not have any abstract methods. These classes lie between interface structures and the common class structures. You may again be thinking about the use of such classes, especially in relation to OOP programming.

They are created to work as the parent shell for derived classes, which will then contain the objects that will be actually processed during subsequent codes and calls in the program. This process is further shown by the following example:

```
package java11.fundamentals.chapter12;
public abstract class Animals {
    public void PrintInfo() {
        System.out.println(GetSound());
    }
    protected abstract String GetSound();
}
```

This shows that we have one defined method in the abstract class, while there is another abstract method of `GetSound()`. Our choice of not instantiating this method means that it can hold different forms based on the creation of descending classes, which may use it differently especially with the use of code overriding.

QUICK CHALLENGE

Should you use interfaces over abstract classes? Give reasons for your answer.

12.2.2.2 Real-World Understanding

The principle of abstraction is best understood by understanding the principle of driving a car. Let us consider that it is a company car, where the company director is authorized to use it in any intended manner. He only needs to “call” the car driving function by contacting the relevant driver and does not require any specific details of the car. On the other hand, the driver is responsible for carrying out the functions required for performing the actual driving.

This means that there are two types of features. The first are the properties or the attributes. The director must know about them, such as the registration number of a car and the name of the driver tasked with driving it. The set of information may include other specifying details such as the color of the vehicle and its comfort level.

The second type of feature is the functions that the called car can perform. It can start from a particular office and take the company employees to a worksite. This may include driving, refueling the car, and other intended functions that are termed as methods. The controlling director, which works here as the “calling code” does not need to know about how they will be performed. He will simply order them to be executed to achieve the required functionality.

This example represents abstraction at its best, where we describe how OOP languages work for programmers, developers, and end users. OOP principles such as abstraction certainly makes it easier to simplify complex tasks and break them according to the required problem-solving steps.

12.2.2.3 Benefits of Abstraction

Abstraction offers several benefits and therefore, it forms the basis of all OOP languages. Here are some excellent advantages of using this specific principle:

1. It creates a barrier, which protects the implementation layer from the code users.

2. It also offers flexibility in terms of later changing the way implementation is carried out. This may be done to improve the coupling structure to loosen it up. A loose system is beneficial, as it allows all involved parties to make the best use of a working contract created through an application interface.
3. It makes it easier to perform debugging and find out which working layer is at fault for a particular problem.
4. It can be delivered through interfaces, allowing the easy correction of code use by an end user or the identification of wrong implementation scheme placed by the developer.
5. It also allows to set up for a divide and conquer policy for breaking a large program into smaller sections, which are easier to correct and implement by setting up interfaces that connect different parts of the complex program.



Does abstraction restrict the functionality of the implementing class?

12.2.3 Inheritance

This is a principle that is specifically designed to improve the performance of structured programming languages. However, these languages produce a lot of duplicate code in a long program, since many code structures are often required in various places. The OOP principle of inheritance solves this problem by creating hierachal sets of coding elements.

There are special classes that have the ability to copy the behaviors and attributes of their specialized functions. You can then only override the specific elements, which are required to be changed in different parts of the program. Your code becomes optimized, as each time the class object is called, a new specialization creates child classes and objects that only have the change in some of the elements.

The main class is termed as the parent class, which contains the source code. Each specialization results in a child class, each with its own set of altered parameters. This creates a situation where all copies will continue to progress in an independent manner. Take the example of classes that handle monetary values and strings as a package. Their children classes may include one that handles the salary information of employees, while one may store the data of the attendees at a function.

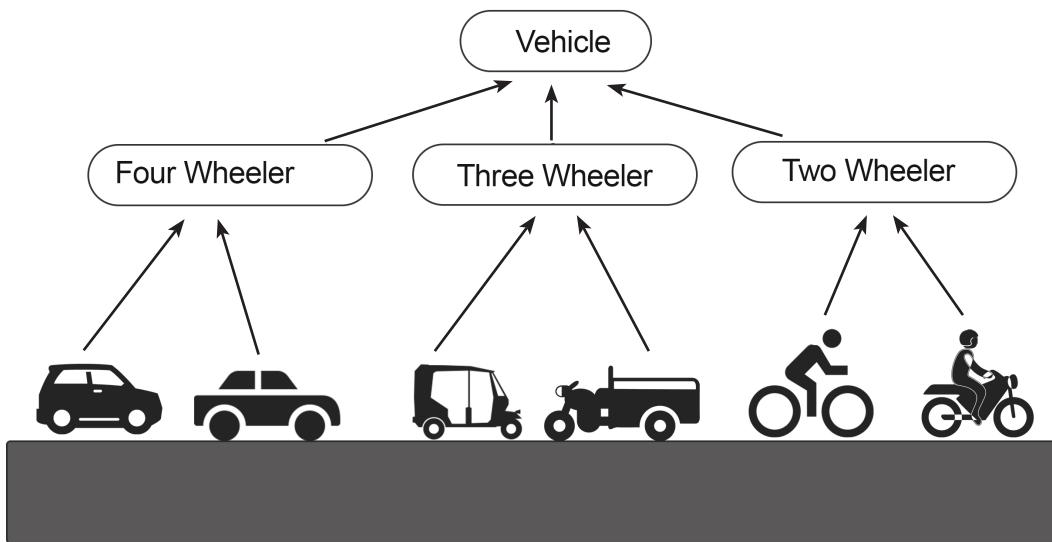


Figure 12.3 Example of Inheritance.

The following example gives a good idea of how inheritance is structured. In Figure 12.3, Vehicle is the superclass, which is also known as base class, and the others are subclasses of Vehicle. These subclasses inherit properties from the superclass Vehicle.

Remember, inheritance can be employed to set up a hierarchy of classes. Each addition only requires setting up new variables and data elements, while the basic set remains available to all the inherited classes. Inheritance simplifies the code and eliminates the instances where we may need to create the same data objects multiple times.

Now, let us see another example, where we create the basic class of a Tutor. We then create various extensions of the class to represent different tutors, like having a ScienceTutor and a LanguageTutor. These two extensions can be created as inherited classes, and only contain new variables that are important for a particular subclass extension. Here is a typical Java example for this situation:

```
package java11.fundamentals.chapter12;
class Tutor {
    String designate = "Tutor";
    String academyName = "NewAcademy";
    void performs() {
        System.out.println("Tutoring");
    }
}
public class ScienceTutor extends Tutor {
    String subject = "Science";
    public static void main(String args[]) {
        ScienceTutor obj = new ScienceTutor();
        System.out.println(obj.academyName);
        System.out.println(obj.designate);
        System.out.println(obj.subject);
        obj.performs();
    }
}
```

The output of the above program is shown below.



```
NewAcademy
Tutor
Science
Tutorial
```

This is an example where we first define the Tutor class that has two initial variables, which is the name designation and the academy name. Each inherited class can then add its own variables that will only be initiated for the members of that class and will not hold value for other objects of the Tutor class. However, we can block a subclass from using the members and methods of the original class if we define them as “private” during their initial declaration.

We can use the same technique to implement a set of well-defined classes that are present in a single hierarchy. This may not hold many levels when discussing people, but it is possible to have several subclasses when creating a complex Java program to imitate a complicated task.

It is possible to have different types of interference in Java. The primary inheritance is the single instance where there is a parent class, which gives rise to a child class. This is performed by extending a class to another class. This can be simply termed for two classes A and B, as B extends to A. However, it is also possible to implement multiple levels and create a large hierarchy, where each parent class extends a child and this occurs at least twice.

It is also possible to build a complete hierarchy, where multiple children classes are present and each has a single parent class. This is an excellent way of implementing functions, where we may have a single class structure that gives rise to objects which may also belong to other categories. Another type of inheritance is the seldom used multiple inheritance, where a single class may belong to multiple classes. Fortunately, this function is not available in Java 11.

Java employs the use of constructors when creating subclasses. It automatically creates inheritance, where all objects are constructed from the top to down method. It is possible to use a specific superclass constructor by employing “super” as the keyword. Remember, only one class can be termed as the superclass when creating a hierachal setting class objects.

12.2.3.1 Advantages of Inheritance

There are several advantages of inheritance, which we have already described. Here, we summarize the primary benefits of the OOP principle of inheritance, especially as they are available when employed in the programming environment of Java.

1. It allows us to derive further classes. This creates a reusable model where we never change the existing classes, which improves the software development time required for program executions.
2. The derived classes generate an extended set of properties, which ensures that programmers create dominant objects that are fully capable of performing the required independent tasks with loose connections to each other.
3. Base classes can give rise to multiple derived classes, creating a dynamic hierarchy capable of providing excellent functionality under different conditions.
4. It is possible to create complex inheritance, which offers the benefits of having all the properties present in the base classes. (Not possible in Java.)
5. All common code belongs to the superclass and only needs to be executed and compiled once during a program operation.
6. It is possible to override methods, which is excellent for defining empty method definitions in base classes and then overriding them, according to the specific use cases.
7. It is possible to optimize the code and arrange the required functionality in an enhanced manner. This ensures that the program code is effective and provides better throughput results.



Can a class have multiple inheritances?

12.2.4 Polymorphism

The last principle of object-oriented programming is harder to explain than the previous three principles. It provides the concept that it is possible to have objects that have the same hierachal position, but behave differently on receiving the same message. Their behavior is different in terms of producing an output when intrigued by a calling code. Figure 12.4 gives an example of polymorphism by showing how one interface can be used by various classes so they can have their own implementations. Figure 12.5 shows how methods can be implemented differently.

Polymorphism works by altering the way objects behave to the calling code. This means that they will carry out their functions in a variety of ways, based on their particular nature. This is a concept that provides an advanced way of making use of objects that are present in a language such as Java. This complex topic is used in high-level programming, but we will make sure that we give some examples in this chapter that will help you understand this important concept better.



Figure 12.4 Example of an interface with different implementation classes.



Figure 12.5 Example of Polymorphism by showing different implementations of an interface.

A better understanding is possible when we define polymorphism as the capability of modifying some methods of an object through overriding. This means that particular objects will serve as subclass instruments of their parent class, when they are showing different behavior when employed using the same calling code and instructions. In other words, objects of the subclasses may have different behavior from the objects of the other subclasses derived from the same superclass. The methods of all the subclasses may return different results.

A good example in this regard is to think of a class that defines big cats. We know that there are various big cats out there, and each has modified methods that they use to stalk and catch prey. Now, the calling code is in the form of a command that asks a cat to catch their prey. Each cat (say, lion or jaguar) will use modified methods to carry out this task. In this manner, the same object works as belonging to different derived classes.

We can define two types of polymorphism, according to its implementation. These two types are overriding and overloading. In this section, we will get introduced to these concepts.

- Overriding:** Here, the compiler is responsible for selecting the method that will be executed after the call. The decision occurs each time the code is compiled. This is also termed as *run-time polymorphism*, because the object takes different shapes during the program execution.
- Overloading:** This is when the specific method that is used by the object is determined by the dynamic position and type arrangement of the object. This is defined not during the execution, but is fixed at the time of program compilation, which occurs prior to execution having fixed behavior. It is also termed as *compile-time polymorphism*.

Polymorphism certainly remains an important principle, and it is one that truly separates OOP languages from functional languages that have a fixed code behavior. Take the example of a class which has a method of animal sound. We implement it within a class of AnimalProperties. Now, we know that each animal has a distinct sound. We want to ensure that the method of `animalSound()` returns a different result that depends on which element we are processing.

We will present two kinds of examples according to the two types of polymorphism that we have described in this section. Here is the first type of implementation:

```

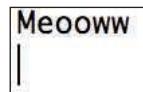
package java11.fundamentals.chapter12;
public class AnimalProperties {
    public void sound() {
        System.out.println("This is top level animal class sound");
    }
}

```

Here are examples of how the same object behaves according to different object behaviors, using runtime polymorphism:

```
package java11.fundamentals.chapter12;
public class Cat extends AnimalProperties {
    @Override
    public void sound() {
        System.out.println("Meooww");
    }
    public static void main(String args[]) {
        AnimalProperties anmObj = new Cat();
        anmObj.sound();
    }
}
```

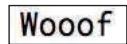
The above example produces the following result. Since we have overridden sound method, it will use the Cat class sound method to generate sound for Cat object.



Here is another way in which the same code would run differently:

```
package java11.fundamentals.chapter12;
public class Dog extends AnimalProperties {
    @Override
    public void sound() {
        System.out.println("Wooof");
    }
    public static void main(String args[]) {
        AnimalProperties anmObj = new Dog();
        anmObj.sound();
    }
}
```

The above program returns the following result.



The first runtime instance will print “Meooww” on the console, while the second one will present “Wooof”. Each behavior is different, although the same class is employed but is designed to have methods that show different behavior according to the required object.

The second example for polymorphism employs the compile time method. This is a technique, where a programmer overloads the required methods to produce distinguished responses. The control for this polymorphism is produced by using the arguments that we are passing to our calling methods. Here is a typical example:

```

package java11.fundamentals.chapter12;
class OverloadingMethod {
    void printOutput(int a) {
        System.out.println("the first number is: " + a);
    }
    void printOutput(int a, int b) {
        System.out.println("The two integers are: " + a + " and " + b);
    }
    double printOutput(double a) {
        System.out.println("The double number is: " + a);
        return a * a;
    }
}
public class OverloadingDemo {
    public static void main(String args[]) {
        double results;
        OverloadingMethod omObj = new OverloadingMethod();
        omObj.printOutput(20);
        omObj.printOutput(20, 30);
        results = omObj.printOutput(2.5);
        System.out.println("The multiplication results is: " + results);
    }
}

```

The above program gives following output.

```

the first number is: 20
The two integers are: 20 and 30
The double number is: 2.5
The multiplication results is: 6.25
|
```

This is an excellent example where we use a single object and keep overloading it in the next instance. The first method of `printOutput()` that runs outputs a single parameter of integer type, which is 20 in our example. The next instance of the method `printOutput()` then takes two integers as arguments, which we set up as 20 and 30. The third instance is when the same object performs a simple multiplication of finding the square of the argument, which is 2.5. The resulting answer is 6.25, which is displayed on the console.

This type of polymorphism is termed as *static polymorphism*. This is because all the shapes that an object can take are already well-defined and take their place during the compilation. There are multiple definitions of the same method. A particular definition is selected based on the argument that we pass on to the method each time we make a call. This means that the method will become static, and will always follow our defined parameters.

This type is certainly great for producing better control over the direction of the methods that can have different instances. A good example is that of a simple calculator, which we only need to produce a fixed quantity of functions. It would be an ideal polymorphic presentation. However, there are certainly other situations as well; we must ensure that the program can behave dynamically to pick out the best course of action.

Dynamic polymorphism ensures that the problem of overriding is only carried out at the runtime stage of the program. This is excellent for controlling methods that may be implemented by using a hierarchy of classes, from our earlier explained principle of inheritance. Let us take the example where we have two classes. The parent class is termed as class First, and the child class is termed as class Second.

There is a method in the child class, which is designed to override the method – say, `exampleMethod()` – that belongs to the parent class. When we assign the child class object in such an example for reference, then this is used to determine which kind of object would be produced at runtime. This means that the type of created object will actually control which version of the same method is called in the Java program.

Remember, it is not important whether the object is now defined as the parent or in the child class. The method is selected based on the specification given to the new instance of the calling class, which will control the flow.

```

package java11.fundamentals.chapter12;
class First {
    public void exampleMethod() {
        System.out.println("This method is to be overridden");
    }
}
public class Second extends First {
    public void exampleMethod() {
        System.out.println("Now overriding the method");
    }
    public static void main(String args[]) {
        First obj = new Second();
        obj.exampleMethod();
    }
}

```

The above program produces the following output:

Now overriding the method

We have created an object that even though it belongs to the first class, uses the construction type of the second class. This will override the initial method and we will get the console out of “Now overriding the method” accordingly. We can mix and match these override situations, but the runtime execution will always select the method according to the type of object that it identifies from the constructor of the object.

QUICK CHALLENGE

Think of any other real-life example of polymorphism and write a program for the same.

12.2.4.1 Advantages of Polymorphism

There are several benefits of the principle of polymorphism. It ensures that programmers can use smarter code design schemes and achieve the following advantages:

1. It ensures that programmers can first fully test and finalize their code before implementing it in a variety of ways. This way, it is possible to always use consistent elements in the final program.
2. Developers do not need to take care of the naming schemes, as polymorphism ensures that the same name can represent different data instances, such as int, double, and other available options in the OOP language.
3. It reduces the present coupling, ensuring that we can create flexible program objects. (We have discussed coupling earlier.)
4. It improves the code efficiency, when a program becomes long with complex functions placed within it, as it ensures that several instances and situations can be adequately handled.
5. Closely related operations become possible by using method overloading, where we can use the same name to designate different methods, each with their own set of parameters.
6. It allows the use of different constructors that can initialize class objects. This ensures that we have program flexibility, with access to multiple initializations.
7. It is carried out during inheritance principle application as well, as general definitions of a superclass can be employed by various objects that add their specific definitions, using the overriding of the available class methods that are not instantiated properly in the superclass.
8. It is excellent for reducing recompilation needs, by ensuring that even sections of a class can have a reusable structure, while the altering requirements may be achieved with the use of polymorphism.



Are there any disadvantages of using polymorphism?

12.3 | Object-Oriented Programming Principles in Application

The OOP principles are important in programming since they offer several advantages. This method is perfect for applying on large programming projects. It ensures that we create reusable code and eliminate redundant elements in our programs. The combining of the running code with the data is perfect for creating self-contained modules that can then employ the advantages offered by modern computers.

OOP principles are excellent for several applications. The most important application is in creating dynamic applications that need updating every few weeks. This is often the case for security programs and ones that help in financial decisions. OOP programs are easily modified as they can have a modular structure. The chances of mistakenly entering wrong logic are also slim in OOP languages. While Java may not be a complete OOP language, it does offer the benefits that are associated with OOP principles.

Another primary application for OOP principles is in creating large-scale programs. These are complex programs that take on many inputs and then perform a number of calculations to produce the desired results. Creating objects allows programmers to set up distinct sections and produce privacy levels. All elements that must remain stable are defined in a private state, while ones that can benefit from an update are set up in a public setting.

Java and other OOP languages are great for creating server/client programs. These programs need efficiency and often require large memory and processing resources. These languages are excellent at creating real-time application solutions, which need to implement reactive programming elements. Parallel programming is best performed in OOP languages that can set up differentiated tasks and define a strategy that provides the ideal scenario for using concurrency during executions.

Artificial intelligence is another avenue for creating OOP programs. Here, we need to make automated improvements to ensure that machine learning becomes a possibility. Any programming application where efficiency and the ability to create smaller modules are required will be best served by using OOP principles and creating reactive programs. These are easily improved while ensuring that the primary legacy elements remain secure and invisible to programmers that may add further functionality.

It is also possible to have bugs in your large programs. OOP languages are excellent for performing the required debugging and the improvement of your code. The use of microbenchmarks can help you identify the performance of various program approaches, ensuring that you avoid logical errors and create a final program version, which is free from a buggy performance.

Since objects often follow inheritance and polymorphism, any errors are easily found. They can often occur due to the misuse of the objects and will seldom require the actual changing in the private object structure, which is another benefit of OOP. With these applications in mind, beginning programs should learn OOP principles and apply them in Java, which also offers primitives along with objects.



Can you use both interface and abstract on a class?

12.3.1 More About Objects

Objects are the primary unit of all OOP languages. They serve as elements that are individual and fully capable of communicating with each other. They can also form hierachal structures and implement logical decisions based on the states and the returns of different operations. Objects are a collection of data types, elements, and their controlling and governing code.

Objects also follow the inheritance principle, where parent objects can give rise to various child objects. The child objects will always be the extensions and contain additional elements that provide a greater functionality over the basic object function and data structure. They are also excellent in terms of only holding the exact information required for their successful performance.

There are several benefits of using objects to employ OOP principles. They allow you to use programming in a way similar to the real-life decisions and understanding of concepts. We define each object by using specific attributes, while also defining a role for it. Programming languages mimic this behavior and provide the functionality of using logic and modularity for designing complex functions during software development.

It allows for realistic control designs that follow the ideal use examples. If we have a gear change system in a car, it always follows the available gears and cannot go beyond the available ratios that are provided by the components present in the transmission. We can implement the same behavior in Java objects, where they only behave in a predictable manner, by only following and taking on values that we have allowed in our object definitions.

Another advantage of using objects in OOP is the use of modularity. We can create independent objects which have the ability to provide a dedicated functionality in a variety of settings. The information that defines the objects is hidden from any instance of the calling codes. Therefore, it works perfectly for creating modules that can interact with each other but not produce any undesirable effects on each other during program executions.

Remember, Java objects work under the system of classes. It provides them an external structure. All Java libraries are a collection of classes, with several methods provide functionality by implementing reusable code, which has already been prepared by an expert for optimized Java functionality.

12.3.2 Classes and Object-Oriented Programming Principles

Classes are important in OOP languages like Java. This is because they allow programmers to use the same kind of logic that we use in the real world to define objects and classify them based on their properties and the function that they can perform.

Let us take the example of cars. There are different types of cars, but all of them can be defined as modes of transportation that use fuel to power locomotion. Your car can be simply defined as one of the instances of the class of cars. It may have certain properties and characteristics that are present in other cars. However, it will never show a property that makes it functionally different from another car. It can have additional behavior though, such as inclusion of a refrigerator and additional entertainment equipment. Figure 12.6 shows the difference between Classes and Objects.

The OOP principles are best acknowledged and used in programming when we employ blueprints. Classes work by providing an important structure, which then allows programmers to build up additional facility and produce improved programming results.

Class definitions only provide the structure and do not have execution element within them. This is provided by the `main()` method, which actually carries out the execution in any Java program. Once you set up these classes that define behavior which may be extended using additional values, it is possible to use the objects that embody the OOP principles that we have described above.

Classes allow the use of the principles of defining every functionality in the form of object instances. With the application of other additional methods, objects can then be used in a modular manner for the intended result. However, programming languages such as Java can go beyond the simple OOP applications and provide additional facilities.

QUICK CHALLENGE

Create a food class and show all the possible child classes of that type. For example, fruits, vegetables, etc.

12.4 | Understanding an Interface

Objects need to interact with the environment around them in the programming application to ensure that they can perform the required interaction. They use methods that expose them and allow them to offer OOP functionality. Objects in Java use the interface to deliver their messages and interact with the running program code. An interface, therefore, is a set of methods which have empty bodies that certain objects can use for their particular function.

A language like Java uses the implementation of different interface elements to show a dedicated and definable behavior. The keyword of “implements” allows the creation of a class object which will implement the empty methods, ensuring that they are employed to allow the object to have a certain formal behavior.

The creation of an interface ensures that the program will only compile when the object uses all the methods that are present in that particular instance. However, this means that they are all successfully defined within that interface element for the successful compilation of the class during a program execution.

Interfaces further simplify the use of classes and objects in Java. An interface provides the answer to the functions of a particular code. Let us again consider the example of a car. A driver can use the steering to ensure that the car remains on the intended part. The steering wheel provides an interface here to use the transparent functionality.

However, the driver may not know at all how the steering process actually works, in terms of the drive axles, power steering mechanism, and the behavior of other mechanical components. The implementation in Java describes these details, which in most applications are redundant and are not required by the user. Interfaces are great to set up when a certain program must deliver a functional process, which is to be used by a normal software program user. This makes interfaces important, especially when employed with Java classes.

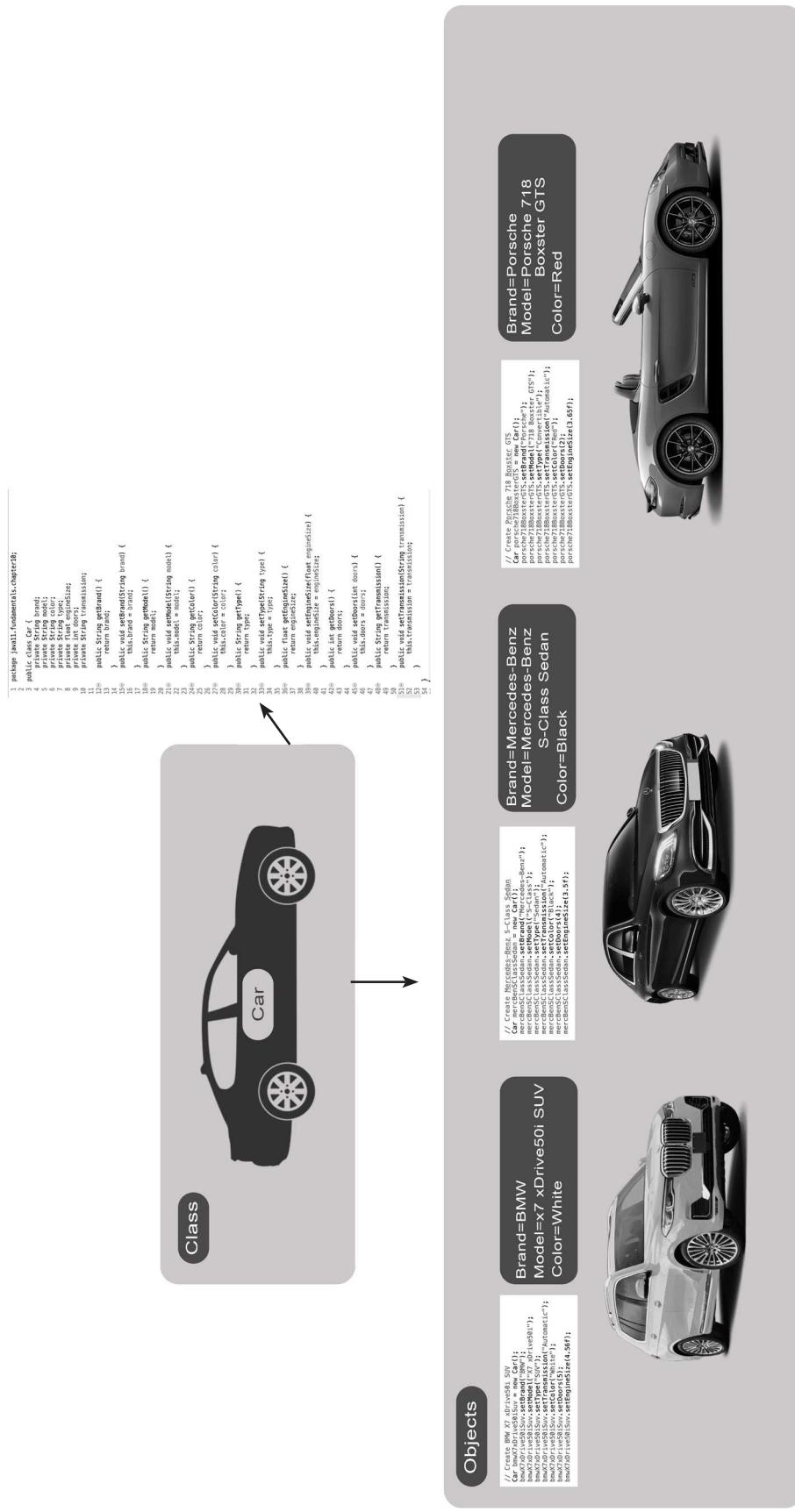


Figure 12.6 Example of Class and Objects.



Can an interface extend multiple interfaces?



12.5 | Overriding and Overloading

There are two important concepts in Java known as overriding and overloading (Figure 12.7). We will explore both these concepts in this section.

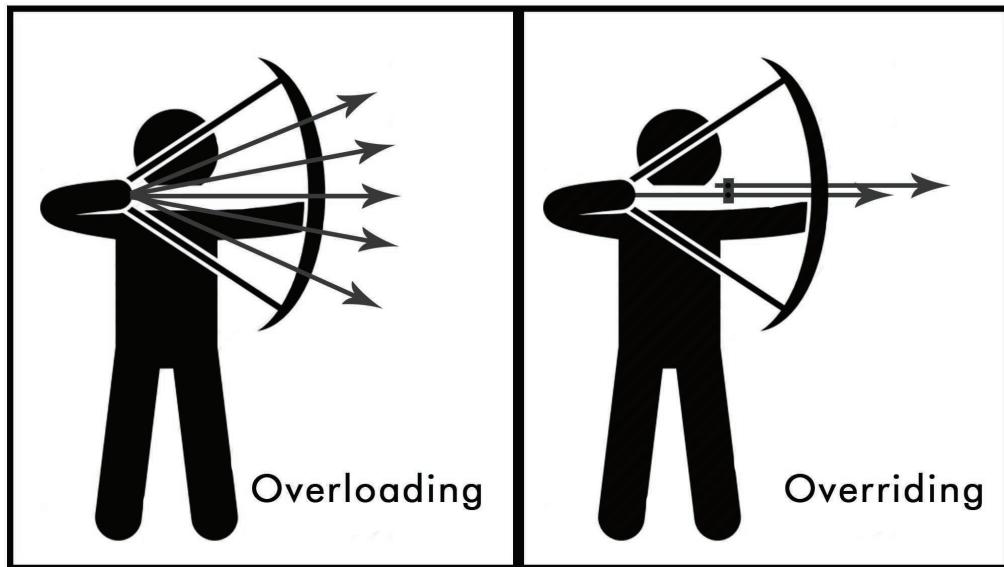


Figure 12.7 Explanation of Overloading and Overriding.

12.5.1 Overriding

Overriding is a process often used in Java programs. You will often create programs where you want to set up methods that you can alter when dealing with different objects or classes that are derived from a superclass. The simplest definition of overriding is that an instance method that occurs in the superclass will override the main method described in the superclass.

One way of using this technique is by writing “@override” annotation before the method. The use of this annotation produces an error from the compiler if it does not find the method first present in the superclass in a dynamic manner. In fact, the compiler actively finds that the method present in the superclass has been overridden in a subclass ahead. If it does not see an overridden implementation of a method definition, it will generate a compiling error.

Remember, overriding is often confused with hiding a static method. Overriding is different since it invokes the method which is described in the subclass. On the other hand, hidden methods can be invoked both from the superclass or the subclass.

In fact, overriding methods in Java offer an important example of an OOP principle. It allows the use of polymorphism. The overriding produces runtime polymorphism, where one object can have different conditions because of the different ways in which the methods can be implemented in the subclass.

Method overriding is easy to use by learning some basic principles.

1. The first rule is obvious, which is to write the exact name of the method as present in the superclass.
2. The second rule is to have the same number of parameters. Remember, a method changes how it behaves but this does not mean that the parameter arrangement can be altered.
3. The third rule for overriding method is the presence of a relationship. The methods can be overridden if they are present in a subclass, where they have an inheritance relationship in the form of the “Is A” model.

Here, we present a simple example of how method override will be employed in the syntax of a Java program. In the program, we present transportation as our superclass and describe the instance of a truck as the subclass, which invokes an overridden method.

```
package java11.fundamentals.chapter12;
public class Transport {
    void run() {
        System.out.println("We use transport vehicles for movement.");
    }
    public static void main(String[] args) {
        Truck obj = new Truck();
        obj.run();
    }
}
class Truck extends Transport {
    void run() {
        System.out.println("We use trucks for transporting loads.");
    }
}
```

This is a simple program, where the `run()` method has two different versions. When we run this method in the `main()`, we employ its second instance, when it will print the message of “We use trucks for transporting loads”.

We use trucks for transporting loads.

This creates a condition where the same program is capable of running different responses based on the OOP principle of polymorphism.

Overriding is important when we have similar objects that we can place in a parent class, but the parameters or formulas that we want to employ in our program may differ. Take the example of banking calculations that will always use the formula, but there may be a change required in the interest rate to ensure that the calculation occurs according to the needs of the specific program object catering to a specific bank.

Remember, we override methods that are dynamic in nature. Dynamic methods are the ones that are called from the objects of a class, rather than the ones that are only described in the parent class. There is another method that is functional in other instances of polymorphism, which is called overloading.

12.5.2 Overloading

The other technique available for changing the methods present in a Java program and then using them for our specific function is the method of overloading. This is a feature where we can use the same name for multiple methods, where the method employed in the program depends on the arguments that we pass during a method’s use. This is a technique where we can use multiple constructors that work on different argument conditions.

This syntax structure available in Java is just like using a constructor to develop the overloading function to have different class extensions. Overloading works by reading the details each time a method is employed in the main program. Take a simple example where we have a multiplication method. This method has two argument instances.

One instance describes the use of two numbers, while the second one is designed to calculate the multiplication of three numbers. Here are the two ways in which they can be run in the code, like having forms of `multi(int num1, int num2)` and `multi(int num1, int num2, int num3)`. We can clearly observe that the only difference between the two method calls is that their argument list is distinct from one another, apart from the actual variables that may be present in the calls.

This type of changing the method is the example of static polymorphism. This is because the type of object required is prepared at the compilation time, when the parameters from a method call are read to understand the required shape of a Java object. Here is an example that describes the use of overloading in Java syntax:

```

package java11.fundamentals.chapter12;
class OverloadMethods {
    public void displaying(char a) {
        System.out.println(a);
    }
    public void displaying(char a, int num) {
        System.out.println(a + " and " + num);
    }
}
public class OverloadExample {
    public static void main(String[] args) {
        OverloadMethods obj = new OverloadMethods();
        obj.displaying('A');
        obj.displaying('A', 100);
    }
}

```

The output of this program is in the form of two lines.

A
A and 100

This is a program which describes the syntax settings of the overload method functionality. We see two separate definitions of the `displaying()` method which both occur in the class initialization. We then create an object of our class, which then employs the variable two times. We only pass a single argument for the first time, which is A. This shows that we are looking for the method which only takes a single character datatype as an argument.

We then employ the method once again, but with two arguments. This automatically lets the compiler know that we are looking for overloading the method with the second definition, which takes as argument one character and one integer. The program automatically now calls this specific method definition and runs the program seamlessly. This is an ideal way of creating programs where you control the number of methods that you have.

You can tweak their definitions whenever you want to perform tasks which are quite similar, by simply presenting different ones in the class definition and separating them with the use of separate set of method arguments for specific use. The arrangement of the arguments, and the type of arguments are all important for providing the directions for the compiler to perform the method overloading. Each time the method definition, which exactly replicates the method call, will be employed in an active Java program.

However, there are scenarios where the Java language syntax is important for controlling the results of method overloading. The datatypes do not have to be fixed when calling the method. The concept of type promotion is employed. This is easily understood with the help of the following example.

If you have a method call with a floating number but the available method definitions only have them for a double integer, then the number will be changed into a double integer. However, this promotion only occurs when the compiler does not find an exact method definition for passing a call in the program code. The rules that govern type promotions in method overloading are:

1. byte to short to int to long
2. float to double
3. int to long to float to double
4. long to float to double
5. short to int to long

Always remember that since method overloading is controlled by having specific and independent set of definition arguments. The compiler will throw an error exception if it finds multiple method definitions that simply take similar arguments. Table 12.1 shows the difference between overloading and overriding methods.

Table 12.1 Overloading vs Overriding

	Overloaded Method	Overridden Method
Argument(s)	Arguments must change in overloaded method	Arguments must not change
Return Type	Return type can change	Return cannot change. The only exception is covariant returns. A covariant return type can be replaced by a “narrower” type in case of overridden method in subclass.
Access	Access modifier can change	Access modifier can be less restrictive but must not be more restrictive. For example, a superclass having a private method can be defined as protected or public in the subclass. But any public method in superclass cannot be declared as private.
Exception	Exception declaration in the subclass can change	Exception declaration can be removed or reduced but must not throw new or wider checked exceptions.
Invocation	At compile time, reference type determines which overloaded version should be selected.	At runtime, object type determines which method should be overridden.

Overloading and overriding are two very important points in OOP. You will be using these two often in your development career. Hence, it is important to master these two concepts.

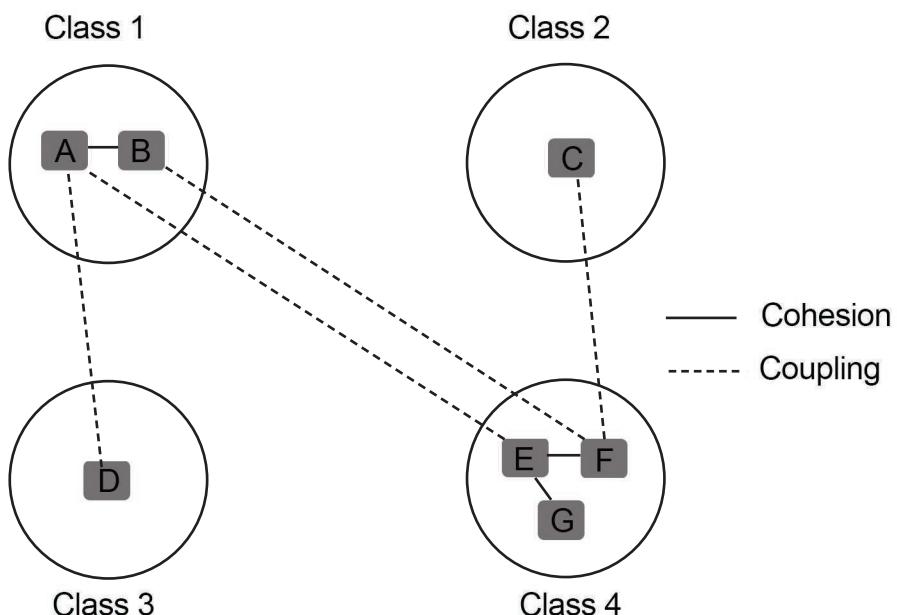
QUICK CHALLENGE

Think of at least five examples where you can apply overloading and overriding principles.

12.6 | Coupling and Cohesion



You will be able to deliver quality code and solve complex problems if you can embrace these two OOP design principles – coupling and cohesion. You should always aim for loose coupling and high cohesion. Let us take a look at what these two terms really mean.



12.6.1 Coupling

Coupling can be defined as the level in which one class has knowledge about another class. If they do not have any common shared knowledge; that is, if one class named A only knows class B via its exposed interface, these two classes can be termed as loosely coupled. On the other hand, a bad design could be if class A relies on parts of class B that are not part of class B's interface. This type of coupling is known as tight coupling, which you must avoid in your design.

12.6.2 Cohesion

Cohesion has to do with the level in which a class is self-contained. A single, well-focused purpose makes the class highly cohesive. There are two major benefits of high cohesive classes.

1. The classes that are highly cohesive are easier to maintain.
2. The well-focused nature of the classes makes them more reusable.

12.7 | Implementation in Java

Java is an excellent programming language and contains many Application Programming Interfaces (APIs), graphics facilities, and other options that are especially great at implementing the OOP functionality. It is a language which now offers the read-eval-print-loop (REPL) system, allowing for quick compiling of code to see how it will behave in a realistic environment.

The core Java features adequately cover all important OOP concepts, which include encapsulation, inheritance, and polymorphism.

Java also offers powerful factory methods that provide much more functional set of objects and classes. This comes with strong interfaces, which can describe any situation and ensure that it is possible to explicitly use code in a manner which ensures that no programmer will be able to use it in a wrong way.

Java is fundamentally a powerful language where you can use all the empowering principles of object-oriented paradigm. However, it still allows you to stay away from situations where you may feel limited by the creation of subclasses and the inheritance implementation structures. It usually imports interfaces and therefore, reduces the instances where we must implement classes in an unintended manner, which often creates further complexities in a Java program.

Creating the interface is important in most Java programs. Interfaces can reduce the number of methods and code that you need to carry out certain functionality. They are often employed with the `@override` annotation, which describes that the method in use is overriding its definitive function, whether it is described from a particular class or a specific interface. Remember, a failed compilation will never change a method in Java.



How many methods can a class have?

12.7.1 Objects Work as Solutions

Java is an excellent language because it allows the use of objects as solutions that resolve well-defined problems through the described parameters. The functions required from a program describe the problem, and then we end using multiple objects and interfaces to carry out the same functionality by using abstraction and employing inheritance at various levels.

Another way in which this situation works is by carrying out problem solving using the available states of objects. We set up the behavior of these attributes in a manner that the presence of the required data ultimately provides the situation, and where the ideal solution to the required solution is identified and achieved.

Most Java programs are created following a specific pattern. First, an industry expert works out the logical solution to a problem. This expert may know about programming or simply provide a human language algorithm or a set of required logical steps. Then, Java program developers use this available information to create an inheritance structure and set up the parameters required to deliver the intended functions through OOP principles.

Java offers multiple avenues for using objects. It allows setting up relationships which are ideal for eliminating code duplications while still ensuring the smart use of available programming avenues and advantages. The major advantage offered by languages such as Java when they first appeared on the horizon was to finally use a programming paradigm, where it was possible to eliminate the repeating code and therefore, create the fountain model of programming.

Object-oriented languages allow programmers to focus on the complex programs, while leaving the emphasis that was often placed at optimizing the code. Java is a language which uses the passing of messages for communicating with the different objects and creating a paradigm where the specifications of the message remain hidden from a common developer using them in a program, thereby creating an efficient working model.

This gives rise to the attributes and the behavior structure, which is shown by class objects in all programming languages. The attributes of an object is set up the way it is defined, especially when using the language to define solutions. The behavior depends on the arguments which are passed to different objects and the methods defined in a class. They employ the various modes of solutions based on the program inputs and outputs.



How many objects can you create in one class?

12.7.2 Ideal Tips for Implementing Basic OOP Concepts

There are several tips which you can follow when implementing OOP principles in your Java programs. Remember, the primary objective of object-oriented languages is to offer swift programming without jeopardizing the security of the prepared programs. The following are the main tips for implementing basic OOP concepts:

1. The first tip is to never repeat the same code in a language like Java. This is a key OOP objective, since we declare all tasks in the form of our required functionality. We can always call the related code accordingly by dividing our data objects in terms of the tasks that we need in a specific program. To avoid this, you should always create a method for a code section that you intend to use multiple times. This will allow you to simply call that method and process the data accordingly in each instance.
2. Encapsulation is the key if you want to continue to improve and alter your code in the future. This is a possible by setting up private methods and variables. You can always change this access to a protected one, which will allow you to always copy the code without ever affecting its primary behavior.
3. Your code should always follow the principle of singular responsibility. Each class that you define and use must cater to a singular need. This means that it will always be easier to call and extend your program classes without affecting the program by a significant margin. Avoid the coupling of key program functions. You can create class objects that combine results from different classes to perform the primary function, but avoid mixing multiple responsibilities in unique code units.
4. Another trick to follow is the use of the open-closed design. You must always keep your classes closed for any modification. This provides data security and ensures that you are following the OOP principles of abstraction and encapsulation. However, your individual methods and specific classes must remain open for further extension in terms of functionality.

This is a great practice to ensure that you can hold on to your successful code, while still allowing your program to embrace new elements that result in improved functionality and scalable program behavior.

You may be wondering how you can achieve the required functionality, if your program is complex and often needs to process large sections of calculations and data elements. The ideal trick to this is to use as many objects that you need. This is the main principle of using an OOP language to your advantage. You create specific instances, call the same methods from the class definition, get the particular task done, and then move on to run other objects in exactly the same manner.

Remember, you do not have to create smart objects that perform complex functions. Use multiple ones and use them within each other for the intended functionality that should always have a singular nature. This allows programmers to truly employ the benefits of using the OOP principles to create an efficient code for any intended function.

5. Another tip is to judiciously use the interfaces that we have described above as special elements that help in OOP implementation. They are by design the helping tools for creating specific objects; use them in a contractual manner. Interfaces should be set up to give hints about the methods that should always be implemented in your source code.

You can work in teams and divide the tasks that you must perform. This is possible if you use specific interfaces that force your employees or coworkers to ensure that they utilize the available methods properly. This creates a coherent code and ensures that all instances of a particular interface always allow the presentation of a unified set of methods.

6. Another ideal tip is the use of dependency injection. This is an excellent technique where the use of a static method or an object is employed for creating the dependencies and important needs of other objects. The dependency is also created using an inject, in which the injection is carried out using the transfer of properties using a client object.

This is an excellent functionality which allows Java developers to implement any type of testing. This technique forces programmers to create database objects which are easier to control using external functionality. This is perfect for testing and ensuring that you can fully obtain the benefits of programming in an OOP language.

7. Another key tip is to compose the inheritance of objects in the right manner. We can use these relationships and the concepts of objects being connected with each other to simply the OOP programming needs. When taken in the context of Java, this means that we should create classes which are simple and can always be extended. This is possible by the strong setting of inheritance and defining the functionality of each in a careful manner.

Remember, an object may be a sub-element of a class, while still having a specific relationship with another object. A good example is that of an apple. An apple can be an object of the fruit class. It can then be described to have hard seeds. The distinction that we need here is that the main class must always be the fruit, as otherwise it will be difficult to define instances where a different set of characteristics is required.

8. Another tip is to create various classes that only connect to each other in a loose manner. If you create objects that are strongly attached to each other, it makes it extremely difficult to reuse your code to its full potential. Create situations where you perform inheritance, but keep the structure flexible enough to always implement additional functionality and the required attributes.

You may have to first study and gather more information about the use of loose classes since the normal concept of inheritances to make a strong attachment between the objects. Remember, creating a code which is flexible and reused plenty of times is far better than creating a strong code which is perfect for use in a typical situation. Consequently, it becomes a burden when used elsewhere in the program because of its dependence on other objects for its optimal functions.

12.8 | Future of Object-Oriented Programming

A key element that we must discuss in this chapter is the future of OOP programming. As we move towards machine learning and the creation of smart code capable of carrying our self-improvements, it is important to consider the future of even the most successful paradigms of today. Here, we present arguments from both sides. We present the bright future ahead as well as describe some pitfalls that are on the horizon for the OOP world.

We first start by presenting a few downfalls. We believe that by knowing them, you can reduce the instances where you face these problems and learn to use OOP programming in the right way, where you become part of the future of OOP programming.

12.8.1 The Downfalls

Inheritance is the key for OOP programming languages. However, it may not be so easy to use in modern, complex programs where it is possible to create large hierachal structures that will all need to be copied in your next program, if you aim to use a child class. This is especially difficult when your class objects also refer to other objects, because you will then require the class hierarchy from all involved objects.

Programmers describe this as the banana problem. If you want to copy a banana from a program code, you will end up copying the banana, its tree, its environment, and the whole jungle just to ensure code consistency through the class structure. This problem appears because of the implicit environment, and computer scientists are already looking at how to control and manage the problems that may appear because of extensive use of class hierarchy structures.

Another problem is produced by encapsulation, which often produces a complex situation during code compilation. A code reference is passed, where it may be required to create a complete and detailed clone object to produce the required level of functionality. Polymorphism is described as an OOP principle, but it can be easily implemented in other ways. In fact, Java allows you to use specific interfaces that produce the same functionality but in a more organized manner.

However, these are common issues and you can work around these problems. The future of the OOP world remains bright, as we describe in the following subsection.

12.8.2 Future Applications

There are various applications that are ripe for use in the future, and are already on the horizon. Take the example of web APIs, which are being prepared with the use of OOP languages such as Java. However, these functions are better prepared using functional programming languages, and current OOP languages also create a shell that produces the same functionality.

One of the best applications of OOP is in the field of user interfaces. User interface (UI) can produce situations which are uncalled for and cannot be resolved by having fixed structures. This is perfect for an OOP language because it is an excellent approach to build the dynamic structure. OOP languages are also great at producing frameworks, as there are several communities out there which provide support for platforms such as the Java Community that supports all Java functions.

Another application is in the field of cloud-based software solutions. Software as a Service (SaaS) is a particular application where OOP languages offer a winning proposition. OOP languages are perfect for creating solutions that can take advantage of the available resources, providing high processing functionality, and using optimum memory resources to produce high impact applications.

There is a strong shift to the cloud and modern data centers that are opening all the time to provide the intended functionality. These centers offer a shared pool of resources and want to provide the ideal service by empowering their hardware resources through the addition of excellent software applications, which can be best prepared using OOP principles, the languages that offer reusable code, and the ability to offer constant updates.

OOP languages can prepare software solutions that can save money for all individuals. Modern data centers need solutions for client/server interfaces, client-side applications, and programs that allow for improved division of the available sources. This is all possible by using a modern OOP platform such as Java. It will continue to get better with a more modernized version already in the pipeline, which helps to provide further support towards the creation of cloud applications.

It is also possible to use languages such as Java with complex platforms and other working elements to create effective situations where it is possible to come up with the ideal results. OOP languages are becoming more powerful, as their discrepancies are reduced by modern computational resources. With the possibility of endless heap size available to the programs, you can produce complex inheritance and create programs that take the full advantage of OOP principles in all facets of the application.

12.9 | Understanding the World

The OOP programming is engaging since it allows developers to equate development ideas to the real-world scenarios and find comparative solutions. There are scenarios like “The Diamond Problem” which are difficult to produce in the OOP environment. However, they represent exceptional cases, where some languages like JavaScript offer a way out for programmers. The Diamond Problem, also known as “Deadly Diamond of Death”, is a multiple inheritance problem. This problem occurs when two classes, say, Movable and Recliner inherit from Furniture, and class Chair inherits from both Movable and Recliner. Now say, equal method is called for Chair and there is no such method in the Chair class but it is in both Movable and Recliner classes. In this case, upon calling equal on Chair, which equal method should get called as Chair class inherits equal method from both Movable and Recliner? This is called “The Diamond Problem”.

Procedural and functional programming paradigms are also helpful. You need to understand when to use OOP to achieve the ideal performance. Using an object-oriented language allows a project manager to divide the development tasks and create sizable chunks of work, where everyone can pitch in without affecting the overall workflow.

A key problem associated with current solutions is that class hierarchies and code supporting elements can often require a large heap size for efficient solutions. With increase in the available resources, it is now easier to implement complex programs that make full use of the important OOP paradigms as independent objects and fully extendable basic classes.

Always remember that we should not force each problem to be resolved using OOP principles. Rather, the ideal way is to start your project management by describing the problem and then work out the possible logical solutions. This will allow you to come up with the relevant OOP ideas in most cases, where you may also learn that some programs may work better using procedural languages.

Summary

OOP is one of several programming paradigms that are in use today. It started out as a programming discipline for creating mainframe computer programs and applications, and became popular gradually. The OOP structure is defined through the principles of encapsulation, abstraction, inheritance, and polymorphism. These principles are incorporated using different tools by various OOP programming languages. Java employs the system of classes and objects to apply these principles.

We have described each of these principles in depth in this chapter. We have presented examples and situations where the OOP paradigm is applied to resolve problems. Another thing that programmers must remember is that each OOP language has its own particular flavor. Some languages cannot be defined as purely object-oriented because they offer other avenues

of creating hierarchy and carrying out a task without creating a strong boundary wall between code implementation and its external behavior.

We also describe the presence of some OOP tools that Java provides. It cannot be termed as a purely OOP language, so we also discussed elements such as an interface, which is present in a different scenario from the implementation of a class object. We described how these principles are placed into action by working out realistic examples.

However, you should remember that this chapter serves as a basic introduction to the OOP world. We have provided some excellent pointers here that you can use to improve your understanding of this programming paradigm.

We completed the chapter by providing an overview of what the future holds for OOP practices. We have identified the key scenarios where OOP languages will continue to excel, while also marking the situations where you should understand the shortfalls of this programming practice.

In this chapter, we have learned answers to the main questions related to OOP:

1. What is object-oriented programming?
2. How does object-oriented programming help create better solutions?
3. What are the principles of object-oriented programming?
4. How do inheritance, polymorphism, encapsulation, and abstraction work?
5. What are the benefits of using abstract classes?
6. How can interface be used to add features to a class?

In Chapter 13, we will learn about generics and collections. We will explore various collection APIs like Map, Set, List, Queue, Collection, SortedSet, SortedMap. And then we will see the concrete implementation classes such as HashMap, Hashtable, TreeMap, LinkedHashMap, HashSet, LinkedHashSet, TreeSet, ArrayList, Vector, LinkedList, and PriorityQueue. We will also learn about Stream API for bulk data operation, forEach, forEachRemaining, removeIf, spliterator, replaceAll(), compute(), merge(), etc.

Multiple-Choice Questions

1. Which one of the following specifiers applies only to the constructors?
 - (a) Public
 - (b) Protected
 - (c) Implicit
 - (d) Explicit
2. Which of the following is not a part of OOP?
 - (a) Multitasking
 - (b) Type Checking
 - (c) Polymorphism
 - (d) Information hiding
3. We use constructors for which of the following?
 - (a) Free memory
4. (b) Initializing a newly created object
 (c) Building a user interface
 (d) Creating a sub class
4. run-time polymorphism is known as _____.
 - (a) Overriding
 - (b) Overloading
 - (c) Dynamic Binding
 - (d) Coupling
5. What does OOPS stand for?
 - (a) Object-Oriented Programming System
 - (b) Object-Oriented Processing System
 - (c) Object-Oriented Programming Structure
 - (d) Object-Oriented Personal Structure

Review Questions

1. What is an abstract class?
2. What is interface? How should we use it?
3. Can you use more than one interface on one particular class?
4. Give a real-life example of polymorphism.
5. How does encapsulation help developers?
6. What is inheritance? Give an example.
7. What is the difference between a class and an object?

Exercises

1. Think of a real-life example of abstraction and draw a diagram. Explain the example.
2. Explain in detail the benefits of inheritance. What would have happened if OOP had missed this principle?

3. Write a program using Eclipse IDE that contains interface and abstract classes. Then create classes that implement these interfaces and extend this abstract class.
4. Explain the concept of overloading and overriding with examples.

Project Idea

Traffic is a day-to-day problem for urban areas. There are various types of vehicles that make it difficult for cyclists and pedestrians. Plan out a software that will allow the city to divide into three types of roads – one for vehicles, one for

cyclists, and one for pedestrians. The rule is that a type cannot use the road that are not intended for its use. Make a detailed diagram for this problem.

Recommended Readings

1. Brett McLaughlin, Gary Pollice, David West. 2006. *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*. O'Reilly Media: Massachusetts
2. Grady Booch, Robert Maksimchuk, Michael Engle, Jim Conallen, Kelli Houston, Ph.D. Young Bobbi. 2007.
3. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley Professional: Massachusetts
3. Lesson – Object-Oriented Programming Concepts: <https://docs.oracle.com/javase/tutorial/java/concepts/index.html>

Generics and Collections

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Generic programming and its application in Java.
- How generics can be employed to improve program functionality and provide room for improvement in the programs.
- Object collections and the functionality that they can offer in Java.
- Important APIs and Java methods that help you use object collections for data manipulation.

13.1 | Introduction

Java is a powerful programming language that facilitates programmers by combining innovative object-oriented functions with legacy options. In this chapter, we focus on a largely ignored part of programming languages, which is the ability to deal with abstract algorithm structures that are later capable of providing the required level of support. Object collections also allow programmers to create and manipulate sets of objects that may offer extended functionality. We will discuss the generics and object collections that can be employed in Java.

This chapter presents a thorough discussion of the implementation classes, application program interface (API), and other options that are available for use in the language, such as mapping and collecting functions. This chapter stresses on the methods that describe the use of generics as well as the important collection schemes that are perfect for use with Java.

We will first establish the objectives of the chapter, then give brief descriptions of the terms of generics and collections in software engineering, and finally discuss the specific functions that you can employ in Java object collections and generic programming methods.

We aim to ensure that this chapter provides the required information for improving your generic programming applications and ensure that you take full advantage of an OOP language that provides object collections for use in complex programming applications.

13.2 | Generic Programming



Generic programming is a key concept in software engineering. It involves the use of algorithms that have the ability to create data types, which can be later specified by instantiating them with a particular set of object or type parameters. This approach has a respectable history and is designed to ensure that it is possible to provide functionality, which can reduce duplication in the program code. Multiple languages employ this functionality, including Java.

Generic programming concept is termed as parametric polymorphism in some programming languages, because it allows the language to have elements and code blocks that can have different forms based on the requirements of a program. Abstract program code is perfect for use in conditions where different programming patterns need to be employed from effectively the same form of code. Generics are also great to ensure that programming errors can be reduced by forcing programmers to use safer methods that allow them to stay away from logical errors.

Java employs generics as a programming facility, which is present in the type system identification in the language. It can be implemented as an object or method type, which is designed to cover operations on different data types, while ensuring that compiling errors are eliminated and a safety net is implemented that takes away logical errors.

Generics in Java have a strong structure which contains type parameters like the following. Variables provide the parameters, which ensures a structure of the class.

```
class class_name<T1, T2, ..., Tn> {}
```

A generic class is capable of declaring multiple type variables, such as Integer and String, in angle brackets (<>). See the following example:

```
class MyClass <Integer, String, Boolean> {}
```

The above code ensures that the class has the required set of parameter types such as Integer, String, and Boolean, which can cover any possible invocation of the class parameters during application.

A generic interface is one that contains type variables, which act as parameters and provide the same interface during the runtime phase of program execution. A generic method is similar to normal Java method, and it can type parameters that may either be following a class object or a Java interface. A generic constructor can work independently in Java apart from its class. This is possible because the Java collections framework is capable of dealing with constructors separately, which may have a parameter list of a generic class or the interface.

13.2.1 Benefits of Generics

Generics have their specific benefits, which suggest that programmers need to use them over other coding schemes. When employed with a programming language such as Java, it allows classes and interfaces to act as parameters and therefore, set up parameters that have greater usability. It is possible to use different inputs and reuse the same code to a greater benefit, which is the main appeal of employing generics.

Java code having generics contains other benefits. Let us take the example of employing data type checks, which occur at the time of compilation. Generics ensure that the compiler can pick up on any errors and generate an error code that describes the missing type safety. This problem is easy to identify and resolve. The same problem goes into the runtime phase if not generics is not used. It becomes a time-consuming exercise to detect, debug, and remove the logical errors that are present in the employed datatypes.

Another benefit of using generics is that it eliminates the use of casting. It is common for the compiler to cast datatypes when adding items in array lists. With the use of generics, the problem is simply eliminated as the required functionality is embedded within the program. The following is an example when casting does not need to be explicitly mentioned:

```
package java11.fundamentals.chapter13;

import java.util.ArrayList;
import java.util.List;

public class Casting {
    public static void main(String args[]) {
        List<String> mylist = new ArrayList<String>();
        mylist.add("I love Arraylist");
        String mystring = mylist.get(0);
        System.out.println(mystring);
    }
}
```

The above program produces the following output.

I love Arraylist

This use of generics ensures that there is no need to perform type casting and change items between primitive data types and the Wrapper classes in Java. Another key benefit of using generics is that it allows Java programmers to develop good habits

and then employ generic frameworks which are type safe, easier to read, and can be customized as and when required during a specific coding application. The main benefits of using generics is to have flexibility in your programs, while ensuring type safety of the objects and methods that you employ.

You can set up structures in your programs that offer improved usability as a complete package. Since you do not specify the fixed data types in your interfaces and generic classes, you can come up with a code that can work with different collections of datasets, such as ones that require a combination of integer and string values. You can avoid the problems that you will face when using non-generics and get a superior code, especially with the use of a single interface definition that offers a strong type control over the employed data objects.

13.2.2 Using Generics in Java

There are various generic elements that you can employ in Java. It is possible to appreciate the benefits of Java if you can compare it with the use of a traditional code to handle the same task. Let us take the example of the Box class for this purpose, where we will describe how it is possible to create a generic version of the class, ensuring that we get excellent benefits when we go ahead with programming and expanding on the created base class.

A simple class will employ private objects where we will have to use two methods of `set()` and `get()` to ensure that the methods employed can use the class object. This does not mean that we can control how the object will be employed during the compilation phase, as it is possible to place a String when the object type is set to work as an integer object in the program code implementation.

This problem is easily resolved when we create a generic version of the same class. A generic version is created using the angle brackets “`< >`”. It ensures that the type declaration that we use employs a variable that we can protect. The following is the code example that shows the use of a generic class, which will offer enhanced type safety:

```
public class Box<T> {
    // where T describes the Type of the object use
    private T t;
    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Here, we can clearly observe that the use of the type “`T`” that we have set up in the generic class declaration replaces all instances of the use of an object in the code. This ensures that the type variable can follow any array, class, another type variable, or any other non-primitive value providing better control and enhanced functionality in the Box class.

There are some conventions that you must follow when employing generic programming. The type parameters are always single capital letters. As it is difficult to use any name without setting up conventions, you may find it difficult to understand generic programs created by other developers. The common letters used by programmers often employ specific parameters that denote the following values:

- 1. E:** It denotes an element and you will find it employed by the Java Collections Framework, which we will define in subsequent sections.
- 2. K:** It denotes key and will be employed for pointing to various data objects and displaying the object information accordingly.
- 3. N:** It stands for number value, where we aim to employ them as parameters for the generic code.
- 4. T:** It denotes type. We can employ a particular type that we want to use in the program and this will offer type safety, which is a characteristic feature of generics in Java.
- 5. V:** It describes value. We can employ the type accordingly when we implement this in a generic class or object.
- 6. S, U, etc.:** These denote that we are employing multiple types. The letters will be denoting them successively, such as S for 2nd and U for 3rd types.

Once you have set up a general class, you need to implement an invocation which needs to present a concrete value. This can include an instance such as that of integer, which is an object. You pass the type argument to your generic class, which is different from using an argument with a class method. This locks up the generics and ensures that it is not possible to employ wrong arguments. Remember, the code will not generate a new object, rather, it will employ the use of a parameter and will produce a reference to your original generic class, which will be of the Integer type in your example:

```
Box <Integer> intBox;
```

Once you have instantiated a generic type in Java, you can continue to work with it in the program, using different type interfaces as well as employ various parameters. We have already described that it is possible to have any number of type parameters in a generic class. This allows us to use objects that may belong to various Wrapper classes, as a common use is to implement functionality for both Integer and String objects.

One way to create a generic interface is to set up an ordered pair of parameters. This can then be set to have keys and values as parameters, allowing the use of the programs in a variety of ways. The use of a generic interface is easier as you do not have to worry as to whether you are using types or objects, as autoboxing in Java will always ensure that it is possible to pass data types to a class, which are automatically converted according to the required use.

It is also possible to use parametrized types that are present within the confines of a generic class definition. Creating such a type is possible in the following manner, if we continue with our earlier example:

```
Box<Integer> intBox = new Box<>();
```

Now, this creates a raw type. This use of code may not be generic in some older versions, but Java fully supports the use of API classes and other detailed generic functionality, which is included in the compiler functionality.

13.2.3 Generic Methods

Now that we have explained the use of generic classes, especially the ones that work with parametrized types, it is time that we describe the generic methods. Creating a generic method is similar to creating a generic type, but the advantage here is that the scope of this type remains limited to a method. It is possible to set up both static and non-static generic methods, while they can also be implemented in the form of class constructions.

If we take the example of a static generic method, we find that the type parameter must be defined before the return type of the method is mentioned in the code. An example for invoking a method for this purpose may be in the form:

```
Pair<String, Integer>pal = new Pair<>("grape", 3);
```

It is also possible to use type inference which allows the use of a generic method as a typical Java method. This means that there is no need to specify the type of parameters in the angle brackets.

Now, we describe the benefit of using a generic method, which is to bound the parameters and ensure only restricted types that can be employed as method arguments. This can be achieved by creating the upper bound for the type and use the “extends” keyword as shown in the following example, like `<U extends Number>`.

```
public class Box<T> {
    private T t;
    public void set(T t) {
        this.t = t;
    }
    public T get() {
        return t;
    }
    public <U extends Number> void inspect(U u){
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }
    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.set(new Integer(15));
        integerBox.inspect("some text"); // error: is presented since this would still be
returning a String
    }
}
```

This is an excellent example of using a method which works with a locked number type. Since we have inserted some string in the inspect method, this directly translates to a String object. This means that the compiler finds that the method cannot work with the available information, since the parameter describes a number extension limiting the parameters that the integerBox can employ. It is also possible to use further complications such as multiple bounds, but we are limiting our use here to the examples that we have presented.

An important concept to understand is that Java is an object-oriented programming (OOP) language. An important principle of OOP is that of inheritance. Inheritance allows creating relationships between different objects and classes. The same case is possible when using generics, where we can set up relations between different elements of generic code.

Creating subtypes of generic classes and interfaces is possible by either extending the class or implementing the interface. The relationship that we create between the various types of classes is possible by using clauses to define it. Here is an example:

```
interface MyList<E,T> extends List<E>{
}
```

This is a simple example where `MyList` is a generic interface which extends the `List` with a single parameter value. Here, it is possible to see that a generic list capable of accepting a string object can have a subtype which can accept an `Integer` object as well as a `String` object. There are wildcards as well that can be employed when setting up generic code. “?” is the most common wildcard and it denotes a currently unknown type. It can be used for a local variable, a field, a parameter value, and a return type.

Generics in Java are designed to check for errors at the compile time. It does not have a use at the time of runtime. The compiler has the function of “type erasure” as an important feature, which eliminates all the type checking code from the final program when creating the byte code. It may cast any type if it is required. The parameterized types do not create any new classes, and therefore, generics in Java do not have any additional overhead runtime resource use.

There are some things that are locked in Generics, such as creating subtypes at a fundamental unit. It is also not possible to create Generic arrays as they will not compile. The idea behind using generics is to create and store specific objects in your Java program. Since the type erasure occurs before the execution phase, the parameters cannot be designed to set up during the runtime phase of the program. This may restrict the programming approach, although it is possible to set up higher parameters such as objects, which still allow the use of subunits such as `Integer` or `String`.

Java Generics are constantly on the rise and developed in its advanced forms. Java 10, which came out in March 2018, has the experimental Project Valhalla present in it. It contains several generics improvements, such as the use of specialized lists and reified generics, which ensure that the actual types can be made available at runtime, even when employing generic code.

Before we can explore the world of Collections, it is important to understand two most important methods that are needed for collections to work properly. These methods are `hashCode()`, and `equals()`. In the following section, we will see why these methods are important and why one should override them.

13.2.4 Overriding `toString()`, `hashCode()`, and `equals()`



Everything in Java other than primitives is an object. Anything you can think of like exception, events, or arrays extends from `java.lang.Object`. Hence, everything contains the methods shown in Table 13.1 that reside in `java.lang.Object`.

Table 13.1 Methods from `java.lang.Object`

Method	Description
<code>boolean equals (Object obj)</code>	This method is used to check if two given objects are meaningfully equivalent
<code>void finalize()</code>	This method is used by the garbage collector when it determines that there are no more references to the object
<code>int hashCode()</code>	Hashtable, HashMap, and HashSet Collection classes use hashing; this method returns hashcode int value for an object
<code>final void notify()</code>	This method is used to wake up a thread which is waiting for the object's lock on which it is used
<code>final void notifyAll ()</code>	This method is used to wake up all the threads that are waiting for the object's lock on which it is used
<code>final void wait()</code>	This method is used to make current thread to wait till other thread class <code>notify()</code> or <code>notifyAll()</code>
<code>String toString()</code>	This returns the text representation of an object

13.2.4.1 Overriding `toString()` Method

This method is useful in giving some meaningful information about the object. If you do not override this method, the object's default `toString()` will be called, which does not give any meaningful information. It only returns the classname followed by @ symbol and the object's hashcode. Let us see the following example:

```
package java11.fundamentals.chapter13;
public class Car {
    public static void main(String args[]) {
        Car car = new Car();
        System.out.println(car.toString());
    }
}
```

The above code gives the following result:

java11.fundamentals.chapter13.Car@6e8dacdf

Similarly, let us see another example of `toString()` in a side-by-side comparison view.

Without <code>toString()</code>	With <code>toString()</code>
<pre>class Student{ int rollno; String name; String city; Student(int rollno, String name, String city){ this.rollno=rollno; this.name=name; this.city=city; } public static void main(String args[]){ Student s1=new Student(101,"Raj","lucknow"); Student s2=new Student(102,"Vijay","ghaziabad"); System.out.println(s1); //compiler writes here s1.toString() System.out.println(s2); //compiler writes here s2.toString() } }</pre> <p>Output:Student@1fee6fc Student@1eed786</p>	<pre>class Student{ int rollno; String name; String city; Student(int rollno, String name, String city){ this.rollno=rollno; this.name=name; this.city=city; } public String toString(){//overriding the toString() method return rollno+" "+name+" "+city; } public static void main(String args[]){ Student s1=new Student(101,"Raj","lucknow"); Student s2=new Student(102,"Vijay", "ghaziabad"); System.out.println(s1); //compiler writes here s1.toString() System.out.println(s2); //compiler writes here s2.toString() } }</pre> <p>Output:101 Raj lucknow 102 Vijay ghaziabad</p>

In the overridden method, you can give some meaningful information about the object, which describes the object for better understanding. For example, if you have a customer class, you can give customer related information, such as name and account number ,so when someone calls `toString()` on that object, it will give useful information about that customer.

```

package java11.fundamentals.chapter13;
public class ToStringExampleWithOverriddenToString {
    public static void main(String args[]) {
        ToStringExampleWithOverriddenToString tse = new ToStringExampleWithOverriddenTo-
String();
        System.out.println(tse.toString());
    }
    public String toString() {
        return "This is an overridden method";
    }
}

```

The above code gives the following result.

This is an overridden method

As you can see, it prints the text which we have provided in the `toString()` method.

13.2.4.2 Overriding `equals()` Method

As you have seen, a simple way to compare two variables is to use `==`. However, `==` is going to check if two references are pointing to the same object or not because `==` simply looks at the bits in the variable, and checks if they are equal or not. If you only care about checking if two object references are equal, feel free to use `==`. However, this is not the case most of the times.

In case of objects, we always want to make sure they are meaningfully equal. Now, `==` can only tell us if these two references pointing to the same object on the heap or not. Hence, we need to override `equals()` method and compare the necessary attributes of the class, which will tell us if they are truly equals. It is also important to make sure which variables you use for the comparisons. If you do not use unique ones per object, you are going to get the wrong result. For example, in case of a Car class, two objects of Car may have similar attributes, such as color, make, and model. If you are using these attributes in the `equals()` method to compare, you may end up concluding that two completely different cars are same, even though they are owned by different persons. Hence, in this case, you may want to compare on a unique element of that class, such as registration number. This will give you the correct result for the comparison.

13.2.4.2.1 Importance of Overriding `equals()` Method

For collections, it is important to override class's `equals()` method because without overriding and giving meaningful equal comparison, it is not possible to find the object in the collections. Hence, we will not be able to use the object of this class as a key in a hashtable or HashMap. Let us explore this in detail. If we do not override `equals()` method, then the Object's `equals()` method will be used instead as everything extends from Object. This Object's `equals()` method uses only `==` operator for comparisons. This `==` comparison means two objects are treated equal if those two references refer to the same object.

Now, let us look at an example where we will not be overriding the `equals()` method. Let us assume that we have two houses that are being sold at the government registry office. Your program is to put house object and owner object in a HashMap, where the house object is the key and owner is the value. Now, when we want to retrieve the information after finishing the process, we need to provide the house object to get the owner's information. This is tricky as we now need to provide the exact reference to the object we used as the key when we added it to the collection. Remember, we cannot create an identical house object again and use it for search. It is because this newly created object will refer to a different location on the Heap, and since Object's `equals()` method is using `==` to compare two objects, it is only comparing the objects' memory locations to check if they are same. Even though these two objects are logically equal as per the `==` comparison, they are different. Hence, we will never be able to create identical object and we will fail to get the data out of HashMap. This is why we will not be able to use an object as a hashtable key or HashMap key without overriding `equals()` method and comparing two objects for logical equivalence.

How do we solve this problem with overriding `equals()` method? We need to make sure we use the suitable attribute to compare two objects so that they can be meaningfully equivalent despite referring to two different objects on the Heap. So, in

our example, let us assume that we can use house address as a comparison attribute in the `equals()` method. Now, we can compare two addresses to make sure these two houses objects are same and hence we can retrieve the owner object from the `HashMap`. However, we can still improve on this a little. Instead of using house object as the key, we can simply use house address as the key so that we will not need to create an identical object reference in the future to retrieve the owner object. We can simply use address String to get the owner object from the `HashMap`. This is possible as String and Wrapper classes work well as keys in hashtables, `HashMaps`, etc., as they override the `equals()` method.

```
package java11.fundamentals.chapter13;
public class EqualsMethodTest {
    public static void main(String[] args) {
        Car myCar = new Car("Mercedez Benz", "S Class", "MH05 12345");
        Car carInGarage = new Car("Mercedez Benz", "S Class", "MH05 12345");
        if (myCar.equals(carInGarage)) {
            System.out.println("Yay!!! This is my Car!");
        }
    }
}
class Car {
    private String brand;
    private String model;
    private String registrationNumber;
    Car(String brand, String model, String registrationNumber) {
        this.brand = brand;
        this.model = model;
        this.registrationNumber = registrationNumber;
    }
    public boolean equals(Object o) {
        if (o instanceof Car) {
            Car car = (Car) o;
            if (car.getBrand() == this.brand && car.getModel() == this.getModel() && car.getRegistrationNumber() == this.getRegistrationNumber()) {
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
    public String getBrand() {
        return brand;
    }
    public void setBrand(String brand) {
        this.brand = brand;
    }
    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
    public String getRegistrationNumber() {
        return registrationNumber;
    }
    public void setRegistrationNumber(String registrationNumber) {
        this.registrationNumber = registrationNumber;
    }
}
```

The above code produces the following result.

Yay!!! This is my Car!

In the above example, class Car overrides `equals()` method which compares car brands, models and registration numbers using `==` operator. Since we are comparing Strings, we can safely use `==` operator for comparison. String is immutable which means string's content will not change once created. And any string objects that we create are stored in the String Constant Pool. This pool cannot have two objects with same content. Hence, if we create another object with the same content, it will only point to the same object in the heap instead of creating a new reference. Therefore, `==` on two strings will give proper result.

13.2.4.2.2 The `equals()` Method Contract

Every class overriding `equals()` method must adhere to the `equals()` contract mentioned below, which is taken from the Java Docs:

1. **Reflexive:** For any reference value `x`, `x.equals(x)` should return true.
2. **Symmetric:** For any reference values `x` and `y`, `x.equals(y)` should return true if and only if `y.equals(x)` returns true.
3. **Transitive:** For any reference values `x`, `y`, and `z`, if `x.equals(y)` returns true and `y.equals(z)` returns true, then `x.equals(z)` must return true.
4. **Consistent:** For any reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return true or consistently return false, provided no information used in equals comparisons on the object is modified.
5. For any non-null reference value `x`, `x.equals(null)` should return false.

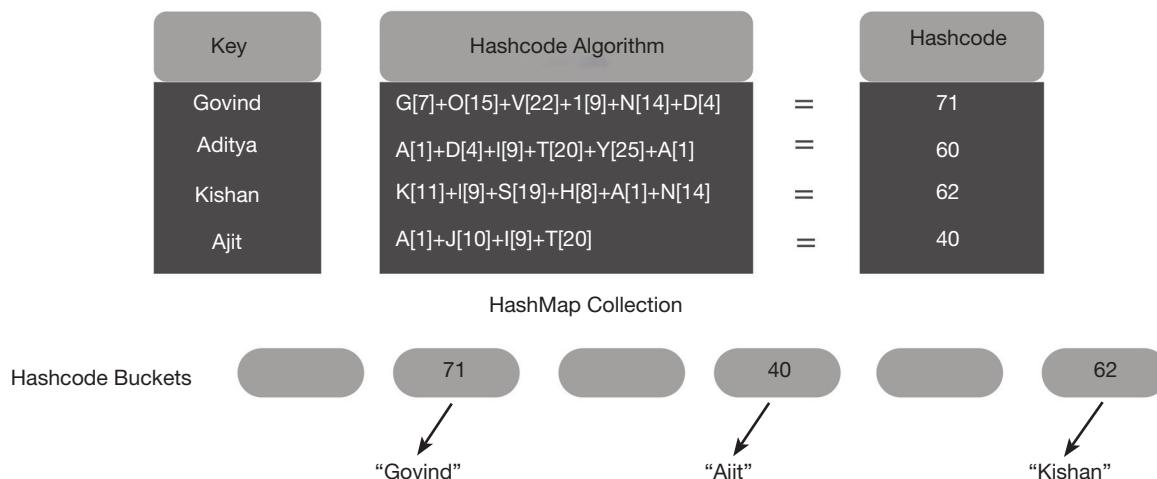
Next, we will look into `hashCode()` method, as `equals()` and `hashCode()` are bound together by a joint contract. This contract states, “if two objects are considered equal using the `equals()` method, they must have identical hashcode values”.

Hence, we need to look into overriding `hashCode()` as well to make sure we create truly meaningful equal objects.

13.2.4.3 Overriding `hashCode()`

In collections, you can think of hashcode as an object ID but not necessarily unique, which is used to increase the performance of large collections of data. Collections like `HashMap` and `HashSet` use the hashcode of an object to see how to store it in the collection and for its retrieval later. Hence, hashcode plays an important role in collections. We need to make sure we have as unique a hashcode as possible to increase the performance of the collections.

Let us try to understand this with the help of an example. In a room, there are 1000 buckets placed, with labels from 1 to 1000. You are given a bunch of slips. Each slip has a name on it and you need to determine a numeric value of each name by adding the letters' numeric equivalents like A=1, B=2, C=3, etc. Each name will give you a number. You need to place this name slip in the bucket that has this number label on it.



Now, if someone comes with a name and asks to get a slip for that name, we can quickly get the number for corresponding to that name and look for the slip in the bucket. This is how collections work. While creating this hashCode generator algorithm, we need to make sure we generate the same hashCode for the same input. So, each time we pass the same name we get the same number back.

As you may have noticed, we may have more than one number for many names that share same letters, such as Ayaan and Nayaan. Since they share the same letters, they will have same number if we add all the characters. It means one bucket may have multiple slips in it. This is acceptable however, if we make our algorithm efficient and we have distributed load in the buckets. This will also speed up the search operation.

Search is a two-step process. First finding the right bucket using `hashCode()` and then searching the bucket for the right element using `equals()` method.

Now, consider the case where our hashCode generator algorithm does not give us consistent result, and we may get different hashcodes at different times by passing the same name. This could be a problem, as when we first added a name in these hash collections, the name would have gone to a bucket with the name's hashCode value. Now, when we try to retrieve the same name, this hashCode generator algorithm gives different hashCode even though the name is same. In this case, we are not going to find the bucket as our name, when we first added is in completely different bucket than we are looking into. Because of this hashCode generator's inconsistent behavior, we will find no-match even though the objects are equal. Hence, it is important to note that for two objects to consider equal, their hashCode must also be equal. This is because if the hashCodes were different, the equal test would fail due to scanning into a wrong bucket.

Let us look at the following example, which shows how to override `hashCode()` method.

```
package java11.fundamentals.chapter13;
public class HashCodeTest {
    public int myVar;
    HashCodeTest(int myVar) {
        this.myVar = myVar;
    }
    public boolean equals(Object o) {
        //We need to check if the object is instanceof of HashCodeTest class. If not we
        can safely return false
        if(o instanceof HashCodeTest) {
            HashCodeTest hTest = (HashCodeTest) o;
            if (hTest.myVar == this.myVar) {
                return true;
            } else {
                return false;
            }
        }else {
            return false;
        }
    }
    public int hashCode() {
        return (myVar * 23); //You can use your own logic. We tried to multiply by a
        prime number.
    }
}
```

13.2.4.3.1 The `hashCode()` Contract

The following contract is taken from the Java API documentation as is. This describes the hashCode contract:

1. Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode()` method must consistently return the same integer, provided no information used in `equals()` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
2. If two objects are equal according to the `equals(object)` method, then calling the `hashCode()` method on each of the two objects must produce the same integer result.

3. It is NOT required that if two objects are unequal according to the `equals(Java.lang.Object)` method, then calling the `hashCode()` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

Now, let us try to interpret this contract (Table 13.2).

Table 13.2 `hashCode()` contract

Condition	Required	Allowed but Not Required
<code>x.equals(y) == true</code>	<code>x.hashCode() == y.hashCode()</code>	
<code>x.hashCode() == y.hashCode()</code>		<code>x.equals(y) == true</code>
<code>x.equals(y) == false</code>		No <code>hashCode()</code> requirements
<code>x.hashCode() != y.hashCode()</code>	<code>x.equals(y) == false</code>	

13.3 | Collections



Object collections are important in Java. They are available as a framework and provide the architecture required to store a collection of objects. The framework and the supporting APIs allow programmers to not only store the collections but carry out various information manipulations. This ensures that it is possible to perform functions, such as String changes and modifications. Figure 13.1 shows Arrays and Collections relationship with Object.

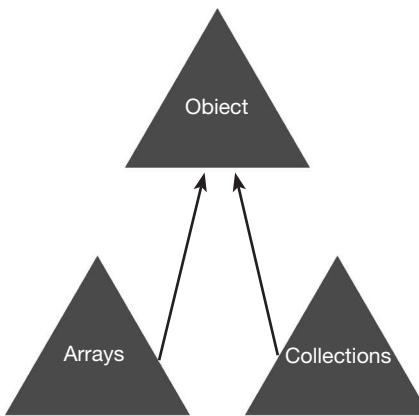


Figure 13.1 Arrays and Collections extends Object.

The use of collections is always optional and allows you to employ the set of objects as a single coherent structure. Usually, the framework contains interfaces and classes to set up the object groups and provide the required functionality to the programmers. This functionality is accessible through the use of collection APIs.

A key point that arises is that arrays also hold primitive objects in Java and therefore, can also be considered as object collections. They are also managed as a singular group item. Thus, it is possible to sort, modify, and work on the data items that are present in an array, as it effectively works as a pointer to the data objects. However, collections are different because their capacity is not assigned when they are instantiated by a programmer.

Collections do not hold the elements of the primitive types. Rather, they hold complete objects belonging to the Primitive Wrapper classes, including Long, Integer, and Double. The earliest Java platform did not have collections implementations, and object manipulation operations had to be carried out using various array options. These were difficult to control and would produce problems during interfacing to have standardized members.

13.3.1 Collections in Java

A concurrency package was developed, which could handle the Java collections. The `java.util.Collection` contains all the collections implementation facilities that we now have available in JDK and IDE software tools for program development. Java architecture combines collections with program generics and offers three types of collections.

1. The first type is the use of ordered lists. These are important when you want to insert objects in a specific order, such as creating a waiting list.
2. The second type is in the form of a dictionary/map that provides a set of information, which can be searched using a reference key to obtain the value of a particular object.
3. The third type is a set. Sets are simply unordered collections. This means that you cannot have similar objects because there is no way to have distinct objects without having an order.

Each of these collection elements has its specific advantages. The ordered lists are excellent as a more flexible way to implement an array. They provide a storage space for their elements that have a better order. Duplication is possible as long as each element is properly placed in its specific position. All positions are available for search, and the listing can be done by either using linking or creating them in the form of an array styled listing.

Another option is the creation of stacks. The stack library in Java allows the use of creating a stack of objects. The stack works on the Last In, First Out (LIFO) principle, when returning objects. This means that the latest item present on the stack will be the first one to be returned, therefore, earning the name of a stack, where you can take out the item on the top first.

There are five operations present that ensure that any vector can be created as a stack. They are as follows:

1. The first method `push()` is used to place any new object in the stack.
2. The second method `pop()` is used to remove an object from the stack.
3. The third method is to look at the top item of the stack.
4. The fourth method finds whether a stack is empty or not.
5. The final method finds a specific item in the stack and describes its positional distance from the top of the stack.

Remember, all stacks are empty when they are created in a Java implementation.

There is also an interface that describes a queue data structure. This is a collection where the items are ordered, but the order remains the one in which they are stored in the stack. All additions are added to the end of the object list, while all removals happen from the front. This is termed as *linked list implementation*. This presents the First In, First Out (FIFO) scheme, which provides a different implementation method, separating it from the LIFO option of the first type of stack.

So, to summarize what we have discussed so far, the following are a few basic operations needed to perform with collections:

1. Adding new objects to the collection.
2. Removing objects from the collection.
3. Looking for objects or group of objects in the collection.
4. Getting an object from the collection without removing it.
5. Looking for a specific object at a specific index.
6. Iterating through the collection one object at a time.

13.3.2 Benefits of Java Collections

The Java Collections Framework offers excellent benefits which are as follows:

1. The main advantage it offers is that it reduces the total programming effort that a developer must go through when creating long and complex programs. This is possible by setting up customized data structures and algorithms like custom collections that can be used as a plumbing system to deliver resources throughout the program.
2. It is important to create a strong data object system in programs. Whether you want to perform integration between various program elements or build operational capability between the different Java APIs, Java Collections provide you the opportunity to create adapter objects. In fact, it is also possible to come up with a conversion code, which allows a Java program to use different APIs and employ them to the optimal advantage of creating efficient and cost-effective programs.
3. The execution speed and the quality of the programming code are important parameters, especially for modern cloud applications. It is possible to increase the program efficiency by using the Collections Framework.

4. When you set up useful data structures in your program, you can come up with multiple interfaces. They are interchangeable because of the use of specific data collections. You have set up data structures that allow you to completely devote yourself to creating better code, while not worrying about how to tackle your objects and data storage elements. The collections allow you to create efficient data structures that can deal with all the information. They are also more flexible than simple arrays that are available in other various programming languages.
5. A key benefit is that unrelated APIs and programming elements can easily work together by using the strong collection interfaces. APIs can interact with each other and share information in the form of the collections that are empowered through the Java Collections Framework. You can employ node names and come up with a strong implementation solution, which is beneficial in a variety of environments.
6. Another advantage is that the use of collections available in Java make it easier to use new APIs and other tools. Naturally, collections can be used to pass data to the APIs as well as obtain the required inputs. This allows programmers to only have the knowledge of collections and then use any API that they require in a particular program. You do not have to learn from scratch every time there is a need to use a new API for a specific Java functionality. In fact, expert Java programmers create their own APIs to help other developers with code simplification and provide support for functions that may be required repetitively in a number of situations. There is no need to reinvent the wheel in terms of sharing objects, data, and information when providing operational capabilities in new APIs. All you need to do is make them compatible with the Java Collections Framework for the required level of connectivity and support.
7. The new data structures that are prepared in your programs can be made to conform to the collection interfaces in Java. This creates a database of objects and information that you can employ later as well. New algorithms can be easily made available for reuse in their implementation if they are prepared according to the needs of the collection interfaces compatible in Java.

13.3.3 Collection Interfaces

Java Collections interfaces form the foundation of the framework. They are often employed with the use of generic coding, which is then combined with a specific data collection. The interfaces that are created are perfect for reducing the runtime errors as all objects are checked during compilation. Java does not set up separate interfaces to ensure that the core collection is easily managed. It throws an exception if an unsupported operation is implemented using the Collections.

The interface here sets up the root of the collection hierarchy. There is a group of objects that make up the collection. The interface allows developers to employ various methods which allow them to check the elements in the collection, as well as perform the functions of removing, adding, and performing an iteration on the available collection elements.

Table 13.3 gives the list of core collection interfaces.

Table 13.3 List of core collection interfaces

Collection	Set	SortedSet
List	Map	SortedMap
Queues	ConcurrentMap	NavigableMap
ConcurrentNavigableMap		

Collections Framework has many collections but not all of them implement the Collection interface. For example, Map interface does not extend Collection interface or none of the classes implement the Collection interface.

The word Collection is little confusing in Java as there are three overloaded uses of the word “collection”. Therefore, you should pay close attention to the word and make sure you refer it properly.

1. **Collection (note lowercase c):** This represents any of the data structures in which objects are stored and iterated over.
2. **Collection (note capital C):** This is the `java.util.Collection` interface. This interface is the one which is extended by List, Set, Queue, etc. Also, note that there is no direct implantation of Collection interface. Only other interfaces extend this interface.
3. **Collections (note “s” at the end and capital C):** This is not an interface but a class from the `util package.java.util`. Collections class. It contains static utility methods to use with collections. For example, `sort`, `emptyList`, `emptySet`, `emptyMap`, `addAll`, `binarySearch`, etc.

In this chapter, we will mainly talk about Collection interface. The following are different aspects of Collection interface:

1. **List:** This Collection interface represents list of things.
2. **Set:** This Collection interface represents unique things.
3. **Map:** This Collection interface represents things with a unique ID.
4. **Queue:** This Collection interface represents things arranged by the order in which they are to be processed.

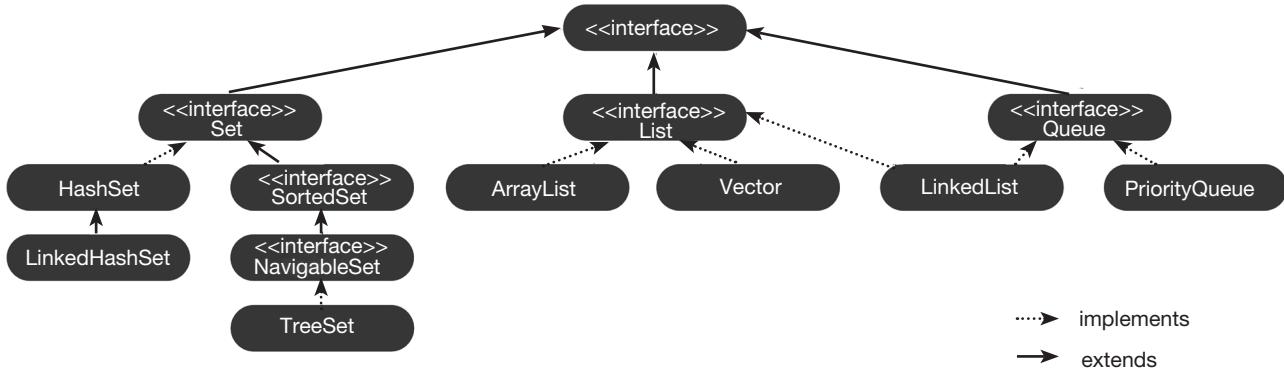


Figure 13.2 Collection Interface.

Figures 13.2 and 13.3 give us a good idea about some of the core interfaces and implantation classes. In Section 13.4, we will explore these classes in detail. The following are the subsets of these interfaces.

1. Sorted
2. Unsorted
3. Ordered
4. Unordered

The implemented classes can be one the following:

1. Unsorted and Unordered
2. Ordered but Unsorted
3. Ordered and Sorted

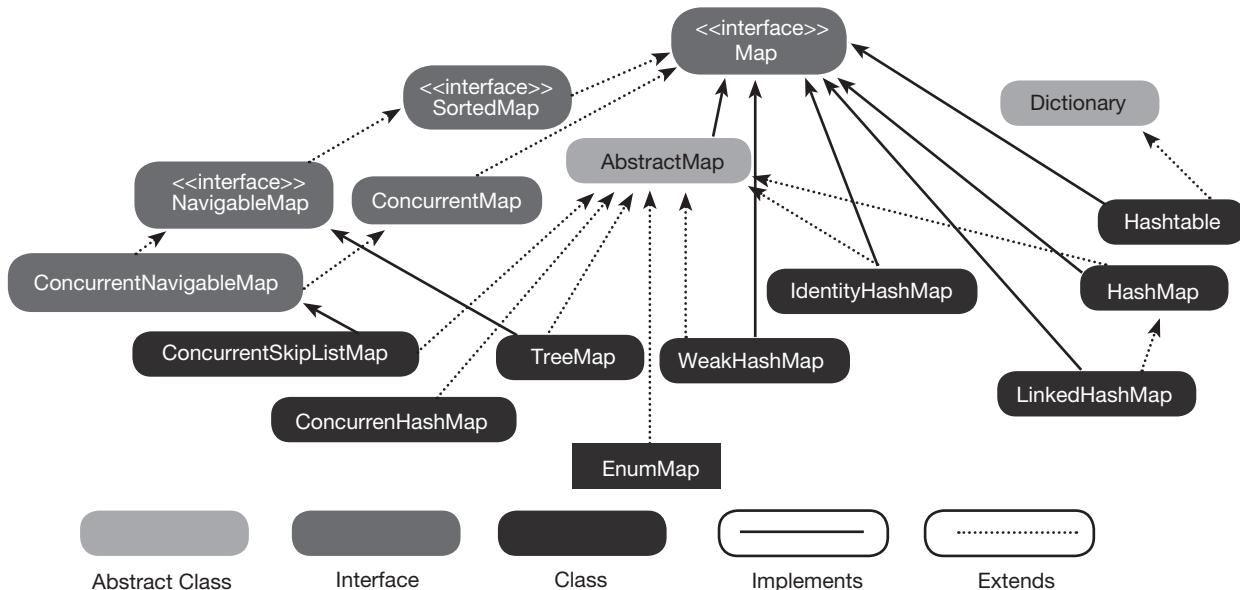


Figure 13.3 Map Interface.

It is important to note that an implemented class cannot be sorted but unordered. As you may have figured, this is because sorting itself is a kind of ordering. Now, let us take the example of HashSet, which is an unordered and unsorted set. On the other hand, the LinkedHashSet is ordered but not sorted. The order of LinkedHashSet is maintained on an insertion basis. The order is set as you add elements to it.

In order for us to understand the concept of collections, first we need to understand the concept of iteration. For a simple “for loop”, we access elements in order of index such as 0, 1, 2, 3, etc. However, iterating over a collection is quite different. Elements are get collected one by one; so there is an order as it starts from the first element. However, in some cases, the first position is not defined the way we think. Take the example of Hashtable, in which there is no first, second, or third order. The elements are simply placed in a chaotic order based on the hashCode of the key. Hence, the first element is determined at the time of iteration because it constantly changes as the collection changes.

Let us explore Ordered and Sorted concepts.

- 1. Ordered collection:** The collection is said to be ordered when iteration can happen in a specific order. As we have seen, Hashtable is not ordered since we cannot determine the order of the elements. The order is determined by the hashCode of the elements and constantly changes as the collection changes. On the other hand, ArrayList is ordered as it inserts the elements on specific indexes. You can determine which index to use to add an element to ArrayList. Hence, it is ordered via index position. However, LinkedHashSet keeps the insertion order; that means that the first element inserted will be on the first order and last element inserted will be on the last order. Some collections use the natural order; that means that they are not only ordered but sorted (by natural order, such as A, B, C, ... and 1, 2, 3, ...). Now, natural order is very straightforward in case of alphabets and numbers, but what about objects? Objects do not have a natural order. So how do we determine the objects’ natural order? For this, Java provides an interface called *Comparable*. This interface has a provision to define a natural order. It means, developers have the freedom to define the natural order for the objects. This rule can be like deciding order based on one of the instance variables, for example, price, age, name, etc. Java also provides another interface called *Comparator*, which can be used to define sort order for objects.
- 2. Sorted collection:** This type of collection is ordered based on some rule(s). These rules are called *sort order*. This sort order is not determined by the time the object gets inserted into the collection, or accessed the last time, or on which position it was inserted. This sorting is based on the properties of the objects that are added to the collection. It means that we simply insert the object into the collection and let the collection determine the order of that element based on the sort order. A collection that keeps an order like ArrayList are not really sorted unless they are explicitly sorted based on some type of sort order. In most cases, the sort order is the natural order of that object.

13.3.4 Collections Classes

Java provides amazing classes that you can employ to carry out the required functions in your program. The main class is the HashSet class. It is backed by the API of HashMap. It does not provide an iteration order and is capable of implementing bucket lists. The initial capacity can be set up in this class, ensuring that it follows a fixed set of capacity increments that save the available program resources. It provides a constant time performance where the elements are dispersed properly in the available buckets.

The TreeSet Class makes use of a TreeMap. The natural order to the creation time is often employed when constructing this Collections class. This class is employed when we want to maintain a consistent structure. It employs comparisons to check different elements, ensuring that it offers the use that we are looking for in a class. Table 13.4 lists some of the top options that are available in Java for use in Collections.

Table 13.4 List of Collections Classes

Map	Set	List	Queue	Utilities
HashMap	HashSet	ArrayList	PriorityQueue	Collections
Hashtable	LinkedHashSet	Vector		Arrays
TreeMap	TreeSet	LinkedList		
LinkedHashMap				

Table 13.5 shows all the classes and indicates whether they are ordered and/or sorted.

Table 13.5 Properties of Collection Classes

Class	Implements	Ordered	Sorted
HashMap	Map	No	No
HashTable	Map	No	No
TreeMap	Map	Sorted	Sorted by natural order. It can also be sorted by custom comparison rules.
LinkedHashMap	Map	Ordered by insertion order. It can also be ordered by last access order.	No
HashSet	Set	No	No
TreeSet	Set	Sorted	Sorted by natural order. It can also be sorted by custom comparison rules.
LinkedHashSet	Set	Ordered by insertion order	No
ArrayList	List	Ordered by index	No
Vector	List	Ordered by index	No
LinkedList	List	Ordered by index	No
PriorityQueue	Queue	Sorted	Sorted by to-do order

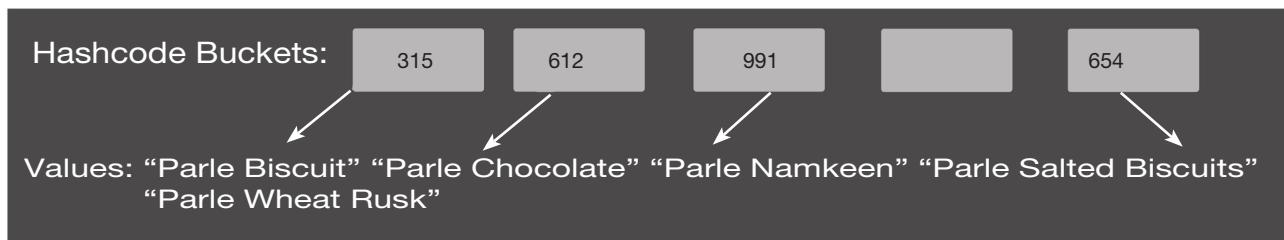
13.4 | Implementing Collection Classes

There are several ways to implement Java Collections functionality. Here, we will explain each concept to build up the desired level of knowledge required for using Collections with Generics. We will describe various API settings that allow the use of collection functionality and provide a detailed discussion on how these concepts are implemented in the language.

13.4.1 Map Interface

Map is a key value pair collection. It takes unique identifier as a key that acts like an ID. Both key and value take object as input. This allows us to search for a value based on the key. Maps and Sets both rely on the `equals()` method to check if the two keys are the same or different.

Figure 13.4 shows how values are stored as key–value pair in Map. Hashcode buckets contain the keys and point to its corresponding values. In the following example, key 315 is associated with value “Parle Biscuit”, 612 is associated with “Parle Chocolate”, etc.

**Figure 13.4** Key–value structure in Map.

13.4.1.1 HashMap

HashMap is a collection class that uses the system of pairs, where one is the key and the other is the corresponding value. The syntax is in the form of `HashMap<K, V>`, where the key comes first while it is followed up by the corresponding value. The objects that are stored in this collection do not have to be ordered as it is employed to find any value by using the corresponding key. It is possible to set up null values in this collection class.

This class is imported from `java.util.HashMap`, which describes its path in the utility library. The following is an example of using this class, in which we have removed the importing of the various other requirements for the program:

```
package java11.fundamentals.chapter13;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class HashMapExample {
    public static void main(String[] args) {
        HashMap<Integer, String> hmap1 = new HashMap<Integer, String>();
        hmap1.put(14, "George");
        hmap1.put(33, "Paul");
        hmap1.put(16, "Jane");
        hmap1.put(7, "Brian");
        hmap1.put(19, "Jack");
        Set set1 = hmap1.entrySet();
        Iterator iterator1 = set1.iterator();
        while (iterator1.hasNext()) {
            Map.Entry ment1 = (Map.Entry) iterator1.next();
            System.out.println("The value is: " + ment1.getValue() + " and key is: " +
ment1.getKey());
        }
        String va = hmap1.get(2);
        System.out.println("Index 2 has value of " + va);
        hmap1.remove(16);
        Set set2 = hmap1.entrySet();
        Iterator iterator2 = set2.iterator();
        while (iterator2.hasNext()) {
            Map.Entry ment2 = (Map.Entry) iterator2.next();
            System.out.println("Now value is " + ment2.getValue() + "and key is: " +
ment2.getKey());
        }
    }
}
```

The above program produces the following output.

The value is: Jane and key is: 16
The value is: Paul and key is: 33
The value is: Jack and key is: 19
The value is: Brian and key is: 7
The value is: George and key is: 14
Index 2 has value of null
Now value is Pauland key is: 33
Now value is Jackand key is: 19
Now value is Brianand key is: 7
Now value is Georgeand key is: 14

This is a detailed program that shows the complete use of the `HashMap` class. All properties are perfectly shown by the use of the example. The first section sets up a `HashMap` collection and places five values with five keys in it. They are displayed using the while conditional loop and follow an alphabetical pattern according to the values, which is possible with the use of the iterator. Next, we remove the value associated with the key of 16 which will remove the "Jane" entry. Now, when we perform

the same operation, we will find the list printed only four values. This time the “Jane” value and its key is not present in the collection.



Can you insert integer as a key and image as value in HashMap?

There are some other excellent methods. The `clear()` method can clear a Map collection completely, while the `clone()` method creates a clone of one map to another, which can then be manipulated while ensuring that we also keep a legacy copy of the original collection. These represent some basic use of this class, allowing us to experiment with this type of collection, where the purpose is to create an unordered pair of object values.

13.4.1.2 Hashtable

Another collection class which is commonly implemented is that of the Hashtable. It is assigned from the `java.util.Hashtable` importing tree. It offers a slightly different implementation than the map, as it creates a table of keys and values, resulting in the production of synchronized set of objects, just like a truth table or a cartesian coordinate system.

It is possible to initiate it using the default constructor or to set up its size. It is also possible to read the elements of a Map in a Hashtable. Considering the earlier example in the HashMap given in Section 13.4.1.1, here is a program that describes the use of a Hashtable in Java:

```
package java11.fundamentals.chapter13;

import java.util.Enumeration;
import java.util.Hashtable;

public class HashtableExample {
    public static void main(String args[]) {
        Enumeration nms;
        String keys;
        Hashtable<String, String> hashtable = new Hashtable<String, String>();
        hashtable.put("Key1", "Adam");
        hashtable.put("Key2", "Brian");
        hashtable.put("Key3", "Charles");
        hashtable.put("Key4", "Dean");
        hashtable.put("Key5", "Peter");
        nms = hashtable.keys();
        while (nms.hasMoreElements()) {
            keys = (String) nms.nextElement();
            System.out.println("Key is " + keys + " & value is " + hashtable.get(keys));
        }
    }
}
```

The above program produces the following output.

Key is Key4 & value is Dean Key is Key3 & value is Charles Key is Key2 & value is Brian Key is Key1 & value is Adam Key is Key5 & value is Peter
--

In the above output, you can see the keys are printed in descending order, from Key4 to Key1, with the corresponding values. The last key entered is always presented at the end. There are various methods that are available for use in this collection class. It is possible to create an enumeration of the keys present in the table, while the `rehash()` method can enhance the size of the table and rehash all its keys. Both keys and the values can be searched within the collection, and it is also possible to get the return of the elements that are present as values in the table.

This class is excellent for use in a variety of conditions. It is perfect when we need to perform collection comparisons to find out something has changed. The table objects can be directly printed by giving the name of the hashtable identifier directly in the print statement.

13.4.1.3 TreeMap

`TreeMap` is a class which implements a navigable map in Java. It employs the natural ordering of the keys that are used to enter values. It is different from the `HashMap` since it creates a map according to the ascending order of its key values. However, this means that this class is not thread-safe and therefore, cannot be properly used with parallel programming in Java, unless it is kept safe.

Here, we present an example to show this behavior as compared to the one that we presented in `HashMap`. This will allow you to understand the use of this collection class, which is perfect in several scenarios where the specific order is important.

```
package java11.fundamentals.chapter13;

import java.util.Set;
import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<Integer, String> trmap = new TreeMap<Integer, String>();
        trmap.put(1, "Object 1");
        trmap.put(17, "Object 2");
        trmap.put(50, "Object 3");
        trmap.put(7, "Object 4");
        trmap.put(3, "Object 5");
        Set set = trmap.entrySet();
        Iterator iterator1 = set.iterator();
        while (iterator1.hasNext()) {
            Map.Entry ment = (Map.Entry) iterator1.next();
            System.out.print("key is: " + ment.getKey() + " and Value is: ");
            System.out.println(ment.getValue());
        }
    }
}
```

The above program produces the following output.

key is: 1 and Value is: Object 1
key is: 3 and Value is: Object 5
key is: 7 and Value is: Object 4
key is: 17 and Value is: Object 2
key is: 50 and Value is: Object 3

This program principles out objects not based on their value and setting. They are printed out with the help of the order of the key values. This means that the TreeMap which is created for the collection automatically comes up with the key order of 1, 3, 7, 17, and 50, which will correspond to their specific values and will be printed in that order.

13.4.1.4 LinkedHashMap

The next implementation class is the LinkedHashMap. It can be presented as a combination of the facilities that are present in a hash table and then combined with the map interface that allows for creating a predictable iteration order for use in Java programs. It is different from a normal HashMap, because it uses a double linked list which connects to the entries that are present in this collections class. The linking defines the way iteration will be performed on the keys that are inserted into the map.

The added functionality provides a situation where we need access the insertion order of the values that become part of the collection. Here is an example to help you understand the use of this Java Collections class:

```
package java11.fundamentals.chapter13;

import java.util.LinkedHashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class LinkedHashMapExample {
    public static void main(String[] args) {
        // Declaring a HashMap
        LinkedHashMap<Integer, String> lihamap = new LinkedHashMap<Integer, String>();
        // Adding the elements to this collection map
        lihamap.put(21, "Abe");
        lihamap.put(35, "Drown");
        lihamap.put(1, "Jack");
        lihamap.put(3, "Karen");
        lihamap.put(100, "Lin");

        // Generating the required set
        Set set1 = lihamap.entrySet();

        // Displaying elements from this collection map
        Iterator iter1 = set1.iterator();
        while (iter1.hasNext()) {
            Map.Entry me = (Map.Entry) iter1.next();
            System.out.print("The key is: " + me.getKey() + " and Value is: " +
            me.getValue() + "\n");
        }
    }
}
```

The above program produces the following output.

The key is: 21 and Value is: Abe The key is: 35 and Value is: Drown The key is: 1 and Value is: Jack The key is: 3 and Value is: Karen The key is: 100 and Value is: Lin
--

The output of this program is different from previous examples. It will return and print the entire list in the order of the way the values are inserted in the program. The first value printed will be Abe with the last value printed will be Lin, in the same order. It is possible now to create chronological lists that you can use for the required functionality in your program. Remember, this map interface requires unique keys and it maintains the insertion order.

There are obvious benefits of using LinkedHashMap. Think of a situation where one module reads a map and copies it, and then produces results that completely depend on the order of the map elements it received.

This map provides access to all the basic operations on collection objects. It may have slower performance than a simple HashMap due to managing the order of the collections. However, performing an iteration is faster since it is only based on the size of the map, as we need access only to the order of the total entries.

Another important element to note is that the LinkedHashMap is not synchronization safe, since a thread can modify the map and affect the order of the stored collections. The knowledge of this type of map is important since it allows programming using Java to make informed decisions and pick the best options from the available set of services.

13.4.2 Set Interface

Set does not allow duplicates. Hence, each element is unique in a set. Similar to Map, Set also uses `equals()` method to check if two objects are equal. In case two identical objects found, only one of them will get inserted into the Set. Let us explore the concept of Set implementation. Figure 13.5 shows the representation of Set.

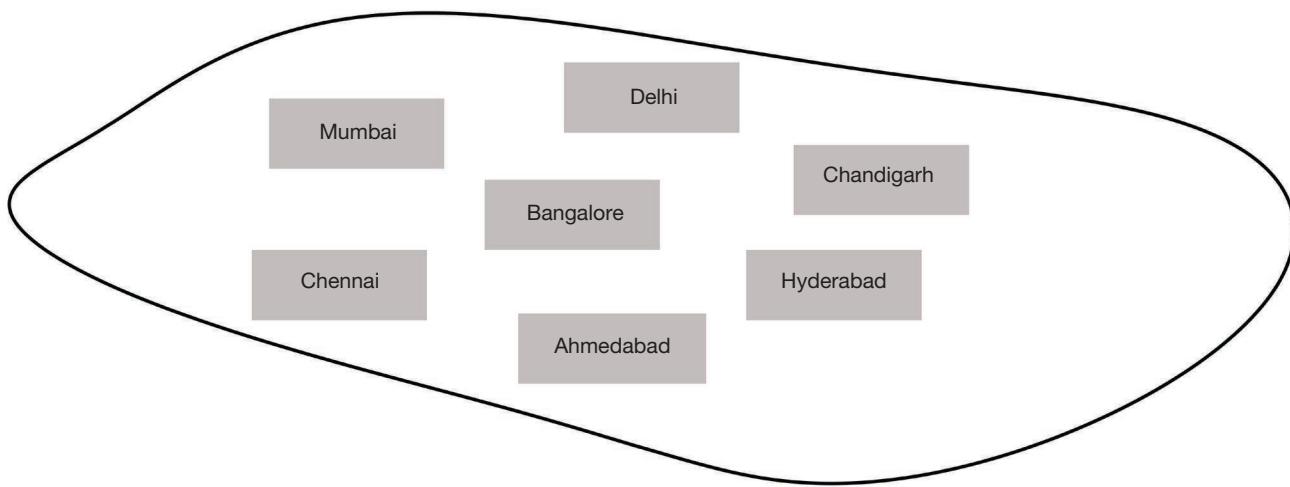


Figure 13.5 Representation of Set.

13.4.2.1 HashSet

Now that we have gone through the classes that employ the map interface, we will discuss the set-based classes that can be implemented from the Set interface in Java. This class does not follow any order; this means that the elements are returned in a random order. It also does not allow the use of duplicate values, as inputting the same value will simply replace the older one.

This set collection also accepts null values, but there can only be one null value stored in a HashSet. The iterator is fail-safe and therefore, it is not possible to modify the set once the iterator has worked on it. However, the remove method within the iterator can remove any value from the hash set without a problem.

The best way to understand the use of a HashSet is to go through a simple example. Here, we describe how programmers can use it as well as display the different items stored in the collection, which are easier to do when compared with the maps that require handling of the keys as well.

```

package java11.fundamentals.chapter13;
import java.util.HashSet;
public class HashSetExample {
    public static void main(String[] args) {
        // Declaring a HashSet
        HashSet<String> haset = new HashSet<String>();
        // Adding different elements including null ones
        haset.add("Apricot");
        haset.add("Mango");
        haset.add("Orange");
        haset.add("Strawberry");
        haset.add("Dates");
        // Adding duplicate elements
        haset.add("Orange");
        haset.add("Mango");
        // Multiple null values
        haset.add(null);
        haset.add(null);
        // Displaying the stored HashSet elements
        System.out.println(haset);
    }
}

```

The above program produces the following output.

[null, Strawberry, Mango, Apricot, Dates, Orange]

The printing of the set will reveal that all values will be displayed without any order. There will only be a single iteration of both the duplicates as well as the null values that we add twice. They are treated similarly, only producing a single record in the HashSet. This is a simple test, which for some can look like the use of an array. However, HashSet implementation offers other advantages by providing methods that allow programmers to manipulate the stored object values.

It is also a good method in terms of the resources that it requires during the operations. It has good performance and works well with the handling of the load that it often possesses. The memory overhead is often not much, and the searching and rehashing operations can use more resources during its application.



Can you get a sorted data from HashSet?

13.4.2.2 LinkedHashSet

Since you have understood the use of a basic HashSet, it is time to discuss the LinkedHashSet class in Java. It also contains unique elements but is different since it maintains the insertion order of the elements added in the class. The sorting of elements are important, especially when a LinkedHashSet must operate with other programming elements, where a change in order can produce a different output altogether.

The double linked list is created for all the elements. It is designed for circumstances where you need to employ an iteration order on the added object values. Using the iterator ensures that the same order is returned in which the elements are added to this hashed set, which has linked elements. Here is an example that shows the use of these practices.

```

package java11.fundamentals.chapter13;
import java.util.LinkedHashSet;
public class LinkedHashSetExample {
    public static void main(String[] args) {
        // Creating a LinkedHashSet for String
        LinkedHashSet<String> lhset1 = new LinkedHashSet<String>();
        // Adding different elements to the LinkedHashSet
        lhset1.add("Z");
        lhset1.add("R");
        lhset1.add("M");
        lhset1.add("O");
        lhset1.add("KKK");
        lhset1.add("EFG");
        System.out.println(lhset1);
        // Now creating a LinkedHashSet for Integer
        LinkedHashSet<Integer> lhset2 = new LinkedHashSet<Integer>();
        // Adding integer elements
        lhset2.add(95);
        lhset2.add(13);
        lhset2.add(0);
        lhset2.add(55);
        lhset2.add(33);
        lhset2.add(61);
        System.out.println(lhset2);
    }
}

```

The above program produces the following output.

[Z, R, M, O, KKK, EFG]
[95, 13, 0, 55, 33, 61]

Printing will start from Z and end with EFG as it was the last item added to the LinkedHashSet. The second LinkedHashSet follows the same method, where it starts from 95 and then continues printing to the last added value of 61. The double linking ensures that it is not possible to insert a duplicate element in this iteration scheme.

It is also possible to remove items. A removal simply updates the iteration, as another system print will find that the item is removed and the item next to it is updated in the order of the storage of the objects present in the set collection. Creating sets are different from the maps as there are no keys that can be used to direct towards specific values.

Linked collection classes use additional CPU resources and require more storage. The simple classes of HashSet and HashMap work well if there is no need to remember the order of the values that are inserted in the program. However, the LinkedHashSet is perfect when you want to maintain the organization of the order of the values, and then set up various objects that may allow you to store, share, and print information sets in your Java programs.

13.4.2.3 TreeSet

As already described with TreeMap, the TreeSet is a class that stores the order of the elements in an ascending order. It also allows you to include null elements, while it is also not synchronized. Although synchronization is possible by explicitly setting it up as a sorted set from the Java Collections Framework. Here is a simple example to show how it works before we discuss some of this class's implementation advantages in specific situations.

```

package java11.fundamentals.chapter13;
import java.util.TreeSet;
public class TreeSetExample {
    public static void main(String args[]) {
        // Creating a String type TreeSet
        TreeSet<String> tset = new TreeSet<String>();
        // Adding various string elements to the above TreeSet
        tset.add("EFG");
        tset.add("Stores");
        tset.add("Tests");
        tset.add("Pens");
        tset.add("Ink");
        tset.add("Jane");
        // Displaying the collection of TreeSet elements
        System.out.println(tset);
    }
}

```

This program will print the strings in an ascending order. The print line will display the following.

[EFG, Ink, Jane, Pens, Stores, Tests]

This shows that printing produces the display in ascending order. This function not only works well with String objects, it is also perfect for use in a variety of Integer objects environments where the order of the collection is of paramount importance. As you can see, it does not allow for repeating values as this will make it impossible to use a sorting scheme.

It is possible to specify the sorting order that you would like to use with this collection class. Also, it cannot hold objects that belong to different primitive types. This means that if a TreeSet has string values in it, it will throw a class exception when a code attempts to add an integer to the collection and vice versa.

Since this tree set can use a comparator to set up the order of the elements, it can be useful in a variety of conditions where the data is randomly collected. However, its use requires that the elements are sorted and are available for use according to a specific order, which is possible by using a comparator when defining the collections class.

13.4.3 List Interface

List is all about index. It offers various methods related to index, such as get(int index), indexOf(Object o), and add(int index, Object obj). List implementations are ordered by index. An object added to List will get added to the end of the list unless you specify a specific index position. Figure 13.6 shows the implementations of List.



Figure 13.6 Implementation of List.

13.4.3.1 ArrayList

The ArrayList is also a part of Java Collections. It is designed to allow programmers to use dynamic array collections. It is obviously slower than using simple arrays, but it is perfect for use when you have to perform multiple array manipulations. It belongs to the AbstractList class and therefore employs the list interface. Figure 13.7 shows the representation of ArrayList.

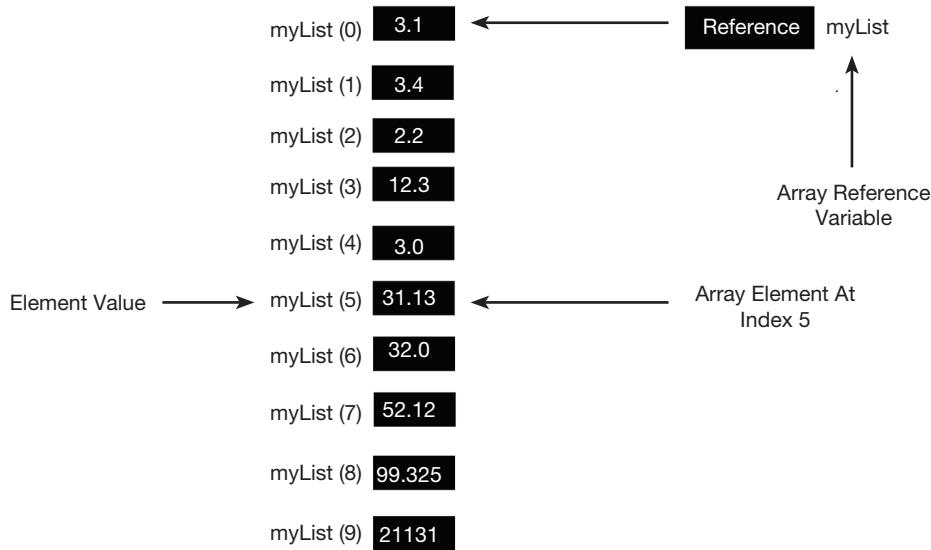


Figure 13.7 Representation of ArrayList.

This class is designed to provide random access to the list of array elements. It cannot be used for primitive types. It is designed for use in the same manner as vectors are used in C++ programming language. It can contain duplicate elements as well, while it also maintains the insertion order of inputted values.

Similar to other collection classes, ArrayList is a non-synchronized class. Random access is made possible because the array operates on the basis of index, which eliminates the need to iterate to reach each item. All object manipulations are slow because there is a strong shifting which occurs when a single element is removed from the ArrayList. This list is easy to create and the available methods allow programmers to add and remove elements, while also ensuring that it is possible to use the index values of the stored array elements.

Here is a program that shows how this class can be implemented in your Java program:

```
package java11.fundamentals.chapter13;
import java.util.*;
import java.io.*;
public class ArrayListExample {
    public static void main(String[] args) {
        // First setting the size of ArrayList
        int size = 5;
        // Now declaring ArrayList with that size
        ArrayList<Integer> arrlist = new ArrayList<Integer>(size);
        // Now Appending new elements in the list
        for (int i = 1; i <= size; i++) {
            arrlist.add(i);
        }
        System.out.println(arrlist);
        arrlist.remove(2);
        // Again displaying the ArrayList after removing
        System.out.println(arrlist);
    }
}
```

This is a simple program, which will first store the first five integers in the ArrayList by the name of “arrlist”. The first print will include [1, 2, 3, 4, 5]. These values are stored sequentially just like a normal one-dimensional array indexing. This means that removing the index 2 will remove the third value from the list, which is 3 in this example. The next print will show the following output.

[1, 2, 3, 4, 5]
[1, 2, 4, 5]

This shows that it is easy to use, manipulate, and set up for different functions. Setting up array lists with fixed initial capacity is the ideal way to go about it. It ensures that you have greater control over your program and can reliably use your lists to create inputs for other program elements.



Is ArrayList safe to use in a multithreaded environment?

13.4.3.2 Vector

The Vector class in Java belongs to the List hierarchy, and works like a growing array of objects. All the stored objects are accessible using an integer index. It is named as a vector as the size of this class varies according to the items that are currently present in the Vector. It provides storage management by setting up a fixed capacity as well as allowing a capacity increment to ensure that it is always possible to store new objects in it.

The advantage of using this class is that it is synchronized and therefore, can be used in all types of Java programs without ever worrying about parallel processing problems. In fact, the dynamic array created in this class often contains legacy methods, which go beyond the scope of the Java Collections Framework.

Vector is great for use, if you are programming for predictable situations where you do not know the size of the required arrays. It is great for situations where you may want arrays that can smartly change their size and always use only the minimal program sources. There are various methods present in this class, which allow programmers to carry out the intended programming. Here is an example:

```
package java11.fundamentals.chapter13;
import java.util.*;
public class VectorExample {
    public static void main(String[] args) {
        // Setting up initial size and increments
        Vector vec = new Vector(3, 2);
        System.out.println("Initial size is: " + vec.size());
        System.out.println("Initial capacity is: " + vec.capacity());
        // Adding elements
        vec.addElement(new Integer(1));
        vec.addElement(new Integer(2));
        vec.addElement(new Integer(3));
        vec.addElement(new Integer(4));
        System.out.println("The capacity after four additions is: " + vec.capacity());
        vec.addElement(new Double(6.55));
        System.out.println("Now capacity is: " + vec.capacity());
        vec.addElement(new Double(5.35));
        vec.addElement(new Integer(8));
        System.out.println("Now capacity is: " + vec.capacity());
        vec.addElement(new Float(9.5));
        vec.addElement(new Integer(10));
        System.out.println("Now capacity is: " + vec.capacity());
        vec.addElement(new Integer(11));
        vec.addElement(new Integer(12));
        System.out.println("First element is: " + (Integer) vec.firstElement());
        System.out.println("Last element is: " + (Integer) vec.lastElement());
        if (vec.contains(new Integer(3))) {
            System.out.println("Vector contains 3.");
        }
        // enumerate the vector elements
        Enumeration vecEnum = vec.elements();
        System.out.println("\nElements in the vector:");
        while (vecEnum.hasMoreElements()) {
            System.out.print(vecEnum.nextElement() + " ");
        }
        System.out.println();
    }
}
```

The above program will produce the following output.

```

Initial size is: 0
Initial capacity is: 3
The capacity after four additions is: 5
Now capacity is: 5
Now capacity is: 7
Now capacity is: 9
First element is: 1
Last element is: 12
Vector contains 3.

Elements in the vector:
1 2 3 4 6.55 5.35 8 9.5 10 11 12

```

This is a detailed program that uses several ways in which the vector is created and employed. We first set up a vector with a capacity of 3 and an increment of 2. However, when we initially find the size of the vector, it is zero due to the absence of any stored value but still showing that it can accept three elements.

We add four elements in the Vector, which will put it above its initial capacity. This invokes an increment of 2, which is shown when the next output of the Vector capacity will show 5 as the number. The capacity is printed two more times, where it will show 7 and 9 with 2-place increments as set up.

We then show that it is possible to find the first and last element that will be stored in the same order in which we have stored them, as there is no hash function performed on them. It is also possible to check for specific elements within the vector using a test performed with the `contains()` method. An enumeration can be employed to read the values from the vector. This can then be displayed on the console.

Vector has its special application when we want to use multiple threads. However, since it is synchronized it can be slow in terms of adding and removing elements, as well performing a vector element search. It is good practice to mention the increment that you want, otherwise the default increment is to double the initial size, which can be wasteful in terms of resources.

13.4.3.3 LinkedList

Java provides another class from the List interface of `LinkedList`. This class provides double linking to store the elements. It can contain duplicate elements, while having the capacity to maintain the insertion order. As we have observed with similar classes, it is not designed for synchronized use. It is quick to manipulate as no object shifting is required with the double linked listing structure. Figure 13.8 shows representation of `LinkedList`.

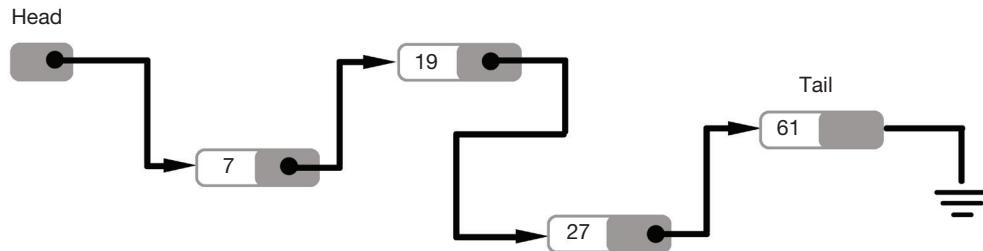


Figure 13.8 Representation of `LinkedList`.

`LinkedList` also implements the `Deque` interface and therefore, it can be used in a variety of ways. It is possible to use this class for stacking or creating a queue of objects. With double links, it is possible to remove the items from the start and end of the linked list. Although it is possible to add or remove items in numerous ways, access to the elements is only available in a sequential manner, where the search can occur either in a forward or reverse direction and will take time according to the position of the specific stored element.

`LinkedList` is excellent for manipulating the order of elements. This allows setting up elements for other program sections. Here is an example that shows the details of this use:

```

package java11.fundamentals.chapter13;
import java.util.LinkedList;
public class LinkedListExample {
    public static void main(String[] args) {
        LinkedList<String> obj = new LinkedList<String>();
        // Adding elements in various orders and positions
        obj.add("a");
        obj.add("b");
        obj.addLast("c");
        obj.addFirst("d");
        obj.add(2, "e");
        obj.add("f");
        obj.add("g");
        System.out.println("Linked list is: " + obj);
        // Now removing elements from the linked list using different options
        obj.remove("b");
        obj.remove(3);
        obj.removeFirst();
        obj.removeLast();
        System.out.println("New linked list after removing: " + obj);
        // Finding elements in the linked list
        boolean stat = obj.contains("e");
        if (stat) {
            System.out.println("List contains the element 'e' ");
        } else {
            System.out.println("List doesn't contain the element 'e'");
        }
        // Other linked list information
        int size = obj.size();
        System.out.println("Size of linked list = " + size);
    }
}

```

The above program produces the following output.

```

Linked list is: [d, a, e, b, c, f, g]
New linked list after removing: [a, e, f]
List contains the element 'e'
Size of linked list = 3

```

This is an excellent example as it shows that we can add to `LinkedList` in a variety of ways. If we add elements normally, they follow their insertion order and are indexed starting from 0. However, as already discussed, it is possible to add at the top or the bottom of the list, which will affect the entire list (e.g., we did that when we added “c” and “d”).

We also added “e” at a particular index, which will move the other items in the list by one position. Similarly, it is also possible to remove the items. We get much greater control when dealing with `LinkedList`. We can use the element value or its index position to look for the element. We can also remove the first or the last element in the `LinkedList`. The size of this list will be according to the items that we have currently stored in this collections class.

13.4.4 Queue Interface

Queue, as the name suggests, holds things that need to be processed in sequence, such as FIFO or LIFO. This structure can be useful for to-do types of list. Queues offers special methods to add, remove, and review elements in addition to all the standard Collection methods. Let us see the implementation class of Queue.

13.4.4.1 PriorityQueue

PriorityQueue is a class that belongs to the Queue implementation, which is known for following the FIFO model. However, there are times where we need to perform processes or prioritize the elements. This is possible by using the PriorityQueue Class, which allows developers to set up priority processing operations.

This class is perfect for creating programs that may have to serve based on the priority of the available information. An example would be in a financial situation, where a company wants its premium customers to receive their processed information first. The PriorityQueue class implementation in Java will work perfectly in such collection scenarios.

This class does not allow for null values, as they are not in line with the function of this collection. It also does not allow you to create objects that cannot be compared with each other since this is essential for setting up the priority. This class uses the natural order of the elements, or employs the comparator that you have set up for the queue.

The smallest element will always be the head (the first element) of this class according to the required ordering. If there is a tie for a specific position, the class will arbitrarily place the elements, so it is best to avoid ties altogether. The queue retrieval operations also work by accessing the element at the head of the queue. It has methods from several structure trees including ones from Object, Collection, AbstractCollection, and AbstractQueue. This class is not thread-safe, but a multithreading version of the class is available for use these days. Here is an example to show how this Java Collection class is used:

```
package java11.fundamentals.chapter13;
import java.util.*;
public class PriorityQueueExample {
    public static void main(String[] args) {
        // Creating the empty priority queue
        PriorityQueue<String> prQueue = new PriorityQueue<String>();
        // Now adding the items
        prQueue.add("C");
        prQueue.add("Java");
        prQueue.add("Python");
        prQueue.add("C++");
        // Printing the most priority element
        System.out.println("The head value by using peek function is: " + prQueue.peek());
        // Now printing all elements
        System.out.println("The total queue elements are:");
        Iterator itr1 = prQueue.iterator();
        while (itr1.hasNext()){
            System.out.println(itr1.next());
        }
        // Now removing the top priority element (or head of queue)
        // And printing the modified PriorityQueue
        prQueue.poll();
        System.out.println("After removing an element with poll function: ");
        Iterator<String> itr2 = prQueue.iterator();
        while (itr2.hasNext()) {
            System.out.println(itr2.next());
        }
        // Removing one value of Java
        prQueue.remove("Java");
        System.out.println("after removing Java with remove function:");
        Iterator<String> itr3 = prQueue.iterator();
        while (itr3.hasNext()){
            System.out.println(itr3.next());
        }
        // Checking a particular element in the PriorityQueue
        boolean a = prQueue.contains("C");
        System.out.println("Does this priority queue contains C: " + a);
    }
}
```

The above program produces the following output.

```
The head value by using peek function is: C
The total queue elements are:
C
C++
Python
Java
After removing an element with poll function:
C++
Java
Python
after removing Java with remove function:
C++
Python
Does this priority queue contains C: false
```

Here, we set up a priority queue which holds the names of the programming languages. Then, we remove an element from the head of the list. This means that now the available values are rearranged. We then remove a specific value as well, and then run a search to check for an item to be present in the collection.

There are various situations in which these priorities can be really helpful during their use in the program. They are perfect for implementing logical decisions as well; for example, when requiring a reading of objects from an available collection.

13.4.5 Stream API

Java has an updated Stream API, which has been significantly improved over its previous version. It is designed to help developers improve their set of aggregate operations, which require the use of a sequence of objects. Stream API is a sequence of operations, where each operation can be set up in the stream. It is designed to produce a stream of objects according to the parameters that are set up in a filter element.

Object references are passed on in a stream, which result in producing a total weight of the objects. Stream operations can belong to various primitive types such as integer, long, and double. They use a pipeline that contains a source, which can be a collection of the several classes that we have discussed above, or it may constitute other elements. It is a lazy implementation where computation is only performed as and when required.

The use of stream is quite similar to a collection but is different in terms of their objectives. Collections are designed to improve the operational control required for managing the elements within them. There are terminal operations that are available for use in this API. Stream operations are great in terms of allowing the program to describe its source material and then defining the required computer operations that must be performed on them.

The iterator method is also available for use. It is possible to run queries on stream sources, while they also allow programmers to perform concurrent modifications. Java streams are designed to provide a sequence of elements on which operations can be performed on demand, differentiating them from the collections that are available in the programming language.

Stream provides both operations of pipelining and internal iterations. Here is a simple example of how different objects and collections can be passed to a stream and then work with some operations:

```
Stream.of("a1", "a2", "a3")
    .findFirst()
    .ifPresent(System.out::println);
```

This stream records the array as a list, and then the stream allows it to be captured in terms of the operations that are required, such as bringing out the first stored item. There are several methods that are helpful when employed with the streams in Java. Take the example of forEach, which works like a loop in the programming language.

The `forEach` loop is great at running an iteration of the items that must belong to a particular category or fulfill a specific condition from the group mentioned within the statement. This loop uses an internal iterator and can employ the interface of a `Consumer` for improved functionality. Here is an example:

```
package java11.fundamentals.chapter13;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class StreamExample {

    public static void main(String args[]){
        List<String> names = new ArrayList<>();
        names.add("Harry");
        names.add("Steve");
        names.add("Adams");
        names.add("Chris");
        names.add("Allen");
        names.forEach(new Consumer<String>() {
            public void accept(String name) {
                System.out.println(name);
            }
        });
    }
}
```

The above program produces the following output.

Harry
Steve
Adams
Chris
Allen

This adds various names and then carries out a loop which accepts and prints out the names from the list. This is a convenient loop which we can employ in place of a “for” loop, while it can cater to the different ways in which we can employ collections in the program loops. There are other methods that are available in collections and streams.

The list classes have various useful methods that can be combined when employed with object streams. It is possible to use the `removeIf()` method, which is designed to remove an object only if a certain condition is met. Sorting is also possible. These are all present in the `java.util` library and it is ideal to include `import java.util.*;` to ensure that complete functionality will be possible in a Java snippet or a complete program.

13.5 | List of Key Methods for Arrays and Collections

The methods covered in following subsections are key methods you will be using frequently in your programs. You should remember these methods so that you can find solutions to the problems you may face while using Arrays and Collections.

13.5.1 Arrays (`java.util.Arrays`)

Table 13.6 lists methods that are useful for Arrays. These methods can be used to work on data using Arrays.

Table 13.6 List of Arrays Methods

Method	Description
static List<T[]> asList(T[])	This method is helpful for converting and binding Array to a List.
static int binarySearch(Object[], key)	This method is useful for searching a sorted array for a key. This returns an index of the value.
static int binarySearch(primitive[], key)	
static int binarySearch(T[], key, Comparator)	This method is useful to search a Comparator sorted array for a given value.
static boolean equals(Object[], Object[])	This method is useful to check if two arrays contain equal content or not.
static boolean equals(primitive[], primitive[])	
public static void sort(Object[])	
public static void sort(primitive[])	This method is useful to sort the array on which it is used by natural order.
public static void sort(T[], Comparator)	This method takes Comparator to sort the elements of an array.
public static String toString(Object[])	
public static String toString(primitive[])	This method prints the content of an Array in String form.

13.5.2 Collections (java.util.Collections)

Table 13.7 lists methods that are useful for Collections. These methods can be used to work on Collections.

Table 13.7 List of Collections methods

Method	Description
static int binarySearch(List, key)	
static int binarySearch(List, key, Comparator)	This method is similar to Arrays, where it searches a sorted list for a given key. It then returns an Index. The overloaded version of this accepts Comparator parameter to search and returns an index of the value.
static void reverse(List)	This method is useful to reverse the order of elements in a given List.
static Comparator reverseOrder()	
static Comparator reverseOrder(Comparator)	This method is useful to accept Comparator to reverse the current sort sequence and return this Comparator back.
static void sort(List)	
static void sort(List, Comparator)	This method is useful to sort the given list by natural order. The overloaded method accepts a Comparator to sort.

13.5.3 Key Methods for List, Set, Map, and Queue

Table 13.8 lists key methods you will encounter periodically while working with List, Set, Map, and Queue.

Table 13.8 List of key methods for List, Set, Map, and Queue

Method	List	Set	Map	Description
boolean add(element)	X	X		This method is useful to add an element to a Collection. In case of list, an overloaded method is available which allows to add an element at a specific index.
boolean add(index, element)	X			
boolean contains (object)	X	X		This method is useful for searching a collection for an object in order to find out if the object is present in the collection or not. This is done by returning a boolean value.
boolean containsKey(object key)			X	
boolean containsValue(object value)			X	For Maps, there are two overloaded versions of the method – one which looks for an object based on a key and the other which is based on a value.

Table 13.8 (Continued)

Method	List	Set	Map	Description
object get(index)	X			This method is useful to get an object for a given index. In case of Map, there is an overloaded method which accepts Key to get the object.
object get(key)			X	
int indexOf(object)	X			This method returns the location of the object in a List.
Iterator iterator()	X	X		This method returns iterator for a List or a Set. Please note that Map does not have this method.
Set keyset()			X	This method is useful for Map to return Set of Map's keys.
put(key, value)			X	This method is useful to add a key/value pair to a Map.
remove(index)	X			This method is useful to remove an element for a given index. There is an overloaded method for Set which accepts object as parameter to remove. Map has an overloaded method which accepts key to remove an element from Map for the specified key.
remove(object)	X	X		
remove(key)			X	
int size()	X	X	X	This method returns the number of elements in a collection.
object[] toArray()	X	X		This method returns a collection of elements in an array form.
T[] toArray(T[])				

The following are some of the operations that are possible on Lists in Java:

1. **void add(int index, object o):** As the name of the operation suggests, this method is used to insert elements into the list. It should, however, be noted that even when elements are inserted anywhere in between the list, there is no chance of overwriting since all other elements are shifted to make room for the newly added element in the list. The index in the parameter of the method indicates the location where the new element needs to be added to the list.
2. **boolean addAll(int index, Collection c):** All of the elements present in the Collection, c, being inserted into the parameters of the method will be added to the list for which the method is being invoked. Again, the elements that were already present in the list are shifted to accommodate the elements that are to be added, ensuring that elements are not overwritten. If the list for which the operation is being invoked changes, it will return true. Otherwise, the value will be false.
3. **Object get(int index):** This operation will return the object that is stored at the location of the index that is input as a parameter.
4. **int indexOf(Object obj):** This operation will return the first instance of the instantiation of the object in the list where it is being invoked. In case the object that was input as a parameter was not a part of the list that invoked it, -1 will be returned to the user as an indication that the object does not exist from where it was invoked.
5. **int lastIndexOf(object obj):** Similar to the operation mentioned in point 4, the lastIndexOf() operation returns the last instance of the object of the list where it is being invoked. In case the object does not exist as an element of the list, -1 will be returned to the user as an indication that something is wrong.
6. **ListIterator listIterator():** Whenever this operation is called, an iterator is returned to the beginning of the list for which the operation was being invoked.

13.5.3.1 Comparable Interface

`Collections.sort()` uses the Comparable interface to sort Lists. Similarly, `java.util.Arrays.sort()` uses it to sort arrays of objects. This interface requires the implementing classes to implement the `compareTo()` method. The following example will help you to understand this better.

```

package java11.fundamentals.chapter13;
public class Food implements Comparable{
    private String item;

    @Override
    public int compareTo(Object o) {
        if(o instanceof Food) {
            Food f = (Food) o;
            return item.compareTo(f.getItem());
        }
        return 0;
    }
    public String getItem() {
        return item;
    }
    public void setItem(String item) {
        this.item = item;
    }
}

```

The above class implements Comparable interface and implements `compareTo(Object o)` method to compare two food objects. However, with this plain implementation, we have to check if the incoming object is `instanceof Food` and then we perform `compareTo` on `String`. `String` implements `compareTo` internally so it is easier to just compare on `String` attributes of objects. However, we can improve this by using the Generics version as follows:

```

package java11.fundamentals.chapter13;
public class FoodWithGenerics implements Comparable<Food>{
    private String item;

    @Override
    public int compareTo(Food f) {
        return item.compareTo(f.getItem());
    }

    public String getItem() {
        return item;
    }
    public void setItem(String item) {
        this.item = item;
    }
}

```

In the above example, using Comparable with Generics reduces the code significantly. This also guarantees that the incoming object is definitely of `Food` type.

Table 13.9 shows the return value for `compareTo()`.

Table 13.9 Return value for `compareTo()`

Condition	Result
<code>currentObject < compareToObject</code>	negative
<code>currentObject == compareToObject</code>	zero
<code>currentObject > compareToObject</code>	positive

13.5.3.2 Comparator Interface

As we have seen earlier, there are two sort methods, one is the `Collections.sort()` that takes List, and the other one is the overloaded method that takes a List and Comparator. This interface gives the ability to sort a given collection in many different methods. Another benefit of this interface is that it allows sorting of any class even if we cannot modify it. This is contrary to Comparable, where we need to modify the class to implement the `compareTo()` method. This is a great advantage, as many times we want to sort objects based on multiple methods and we also do not have access to the source its code. Just like Comparable interface, the Comparator interface is also very easy to implement, which offers a single method `compare()`. The following code will help you to understand this better.

```

package java11.fundamentals.chapter13;
import java.util.ArrayList;
import java.util.List;
public class FoodWithComparator{
    private String item;

    public static void main(String args[]) {
        List<Food> junkFoodItems = new ArrayList<Food>();

        //Create a list of Food
        junkFoodItems.add(addItemToFoodList("Pizza"));
        junkFoodItems.add(addItemToFoodList("French Fries"));
        junkFoodItems.add(addItemToFoodList("Milk Shake"));
        junkFoodItems.add(addItemToFoodList("Burger"));
        junkFoodItems.add(addItemToFoodList("Fried Chicken"));

        System.out.println("Before Sorting : " + junkFoodItems.toString());

        //Create an object for Comparator to sort by item
        SortForComparator sfc = new SortForComparator();

        junkFoodItems.sort(sfc);

        System.out.println("After Sorting : " + junkFoodItems.toString());
    }

    public static Food addItemToFoodList(String item) {
        Food f = new Food();
        f.setItem(item);
        return f;
    }

    public String getItem() {
        return item;
    }
    public void setItem(String item) {
        this.item = item;
    }
}

```

The above class uses a custom comparator that we have created as shown below. Also note that we are using the Food class which we have created in the earlier in Comparable example in Section 13.5.3.1.

```
package java11.fundamentals.chapter13;
import java.util.Comparator;
public class SortForComparator implements Comparator<Food>{
    @Override
    public int compare(Food f1, Food f2) {
        return f1.getItem().compareTo(f2.getItem());
    }
}
```

As you can see, we use food item to sort. This method is similar to the `compareTo()` method. In fact, in that method, we used the `compareTo()` method from String implementation to compare two strings. Executing the above program gives the following result.

```
Before Sorting : [java11.fundamentals.chapter13.Food@7a79be86, java11.fundamentals.chapter13.Food@34ce8af7, java11.fundamentals.chapter13.Food@b684286,
java11.fundamentals.chapter13.Food@880ec60, java11.fundamentals.chapter13.Food@3f3afe78]
After Sorting : [java11.fundamentals.chapter13.Food@880ec60, java11.fundamentals.chapter13.Food@34ce8af7, java11.fundamentals.chapter13.Food@3f3afe78,
java11.fundamentals.chapter13.Food@b684286, java11.fundamentals.chapter13.Food@7a79be86]
```

The program produces an interesting output. Although the program has sorted the objects based on the items' natural order, we are not able to see them. Can you guess why we see this instead of Food item names? This is because we have not overridden the `toString()` method and we are using it from the base class Object, which only prints Class name followed by @ and hashCode. You can easily fix this. For a reminder on how to do so, go to the Section 13.2.4.1 "Overriding `toString()` Method".

The following is the updated Food Class with overridden `toString()` method.

```
package java11.fundamentals.chapter13;
public class FoodWithToString implements Comparable{
    private String item;

    @Override
    public int compareTo(Object o) {
        if(o instanceof FoodWithToString) {
            FoodWithToString f = (FoodWithToString) o;
            return item.compareTo(f.getItem());
        }
        return 0;
    }
    public String getItem() {
        return item;
    }
    public void setItem(String item) {
        this.item = item;
    }

    public String toString() {
        return this.item;
    }
}
```

Now, we need to update the Comparator implemented class and program, which is given below.

```

package java11.fundamentals.chapter13;
import java.util.Comparator;
public class SortForComparatorWithFoodWithToString implements
Comparator<FoodWithToString>{
    @Override
    public int compare(FoodWithToString f1, FoodWithToString f2) {
        return f1.getItem().compareTo(f2.getItem());
    }
}

```

And now the program can use this newly created SortForComparatorWithFoodWithToString class for sorting.

```

package java11.fundamentals.chapter13;
import java.util.ArrayList;
import java.util.List;
public class FoodWithComparatorWithFoodToString{
    private String item;

    public static void main(String args[]) {
        List<FoodWithToString> junkFoodItems = new ArrayList<FoodWithToString>();

        //Create a list of FoodWithToString
        junkFoodItems.add(addItemToFoodList("Pizza"));
        junkFoodItems.add(addItemToFoodList("French Fries"));
        junkFoodItems.add(addItemToFoodList("Milk Shake"));
        junkFoodItems.add(addItemToFoodList("Burger"));
        junkFoodItems.add(addItemToFoodList("Fried Chicken"));

        System.out.println("Before Sorting : " + junkFoodItems.toString());

        //Create an object for Comparator to sort by item
        SortForComparatorWithFoodWithToString sfc = new
SortForComparatorWithFoodWithToString();

        junkFoodItems.sort(sfc);

        System.out.println("After Sorting : " + junkFoodItems.toString());
    }

    public static FoodWithToString addItemToFoodList(String item) {
        FoodWithToString f = new FoodWithToString();
        f.setItem(item);
        return f;
    }

    public String getItem() {
        return item;
    }
    public void setItem(String item) {
        this.item = item;
    }
}

```

Let us run this program and analyze the output.

```
Before Sorting : [Pizza, French Fries, Milk Shake, Burger, Fried Chicken]
After Sorting : [Burger, French Fries, Fried Chicken, Milk Shake, Pizza]
```

From the above program, you can see how easy it is to use Comparator to sort collections.

13.5.3.3 Comparable versus Comparator

Table 13.10 shows the difference between these the interfaces Comparable and Comparator.

Table 13.10 Difference between Comparable and Comparator

Comparable	Comparator
int firstObject.compareTo(secondObject) >Returns the following: firstObject < secondObject – Negative firstObject > secondObject – Positive firstObject == secondObject – Zero The Class that needs a sorting must implement Comparable. Hence, it needs to be modified. With this, only One sort sequence is possible. This is implemented by Java's core classes like String, Wrapper, Date, Calendar, etc.	int compare(firstObject, secondObject) >Returns similar values as Comparable The Class that needs a sorting need not be modified. Comparator allows building a separate class to sort desired elements. This class can later be used to sort the elements in any class. Since we are creating a separate class to sort, we can create as many classes as we want. Hence, we can create many sort sequences. This is intended to be used by third-party classes to sort instances.

Summary

Java improves the facilities of Generics and collections that were present in previous versions. It significantly enhances the capabilities of the classes that are available for creating customized set of objects. Programmers need to learn the concept of generic programming.

Generics present in Java provide the facility required for efficient programming. Java offers internal Generics as well as generic methods that allow programmers to improve their coding performance.

Java also provides a detailed collections framework. This framework is designed to allow programmers to create specific collections of objects and pass them around when using different Java APIs and other functional elements. The framework provides various interfaces and describes how these interfaces are often introduced by the collection classes, which are available for implementation in the programming language. We have described these collection classes, which you can use in a variety of applications.

There are various implementations of List, Map, Set, and Queue that all have their advantages when employed for keeping a record of the objects.

We also discussed stream API, which allows for an alternate way of creating useful connections and for using them to get the job done in a variety of environments.

In this chapter, we have learned the following concepts:

1. Generic programming and how to use it in a program.
2. Generic methods and uses.
3. Collections in Java and their benefits.
4. Collection interface and implementation of collection classes.

In Chapter 14, we will explore error handling, logical errors, semantic errors, try-catch-finally block, and checked versus runtime exceptions.

Multiple-Choice Questions

1. Which of the following is the most apt reason for using generics?
 - (a) Generics makes the code faster.
 - (b) Generics are useful for adding stability to the code by making bugs detectable during runtime.
 - (c) Generics are useful for making the code more readable and optimized.
 - (d) Generics are useful for adding stability to the code by making bugs detectable at compile time.
2. Which of the given parameters is utilized for a generic class to return and accept a number?
 - (a) V
 - (b) N
 - (c) T
 - (d) K
3. _____ permits us to invoke a generic method as a normal method.
 - (a) Inner Class
 - (b) Type Interface
 - (c) Interface
 - (d) All of the above
4. _____ consists of all the collection classes.
 - (a) Java.awt
 - (b) Java.util
 - (c) Java.net
 - (d) Java.lang
5. What do you understand by a Collection in Java?
 - (a) A group of classes
 - (b) A group of interfaces
 - (c) A group of objects
 - (d) None of the above

Review Questions

1. How are Generics useful?
2. What are the different collection interfaces?
3. Which collection class is used to store key–value pair data?
4. What is the difference between ArrayList and Vector?
5. Explain stream API.
6. What is LinkedList? How it is useful?
7. What are implementing classes of Map interface?
8. When is Set useful? How do we use it?

Exercises

1. Create a chart to show all the collection interfaces and implementation classes.
2. Write a program that demonstrates the use of all the collection classes.
3. Use a real life example to write a program to implement PriorityQueue.

Project Idea

Create a program to add and display non-persistent data of all the vehicles entered into a parking lot. Non-persistent data means that we cannot use the database to store this data and hence we have to hold this data in memory. The data should be captured in a way that we can segregate vehicle-specific and driver-specific information separately but linked to each

other. Finally, display this information on a web page and the admin should be able to search the data based on various features such as color of the car and registration number.

Hint: Use various collection classes to hold this data in memory.

Recommended Readings

1. Philip Wadler, Maurice Naftalin. 2010. *Java Generics and Collections: Speed up the Java Development Process*. O'Reilly Media: Massachusetts
2. The Java™ Tutorials, Oracle: <https://docs.oracle.com/javase/tutorial/java/generics/index.html>
3. The Java™ Tutorials, Oracle: <https://docs.oracle.com/javase/tutorial/collections/intro/index.html>

Error Handling

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Error handling.
- Logical errors.
- Syntactical errors such as capitalization, string split, missing or improper imports of classes, and missing curly braces.
- Semantic errors such as improper use of operators, incompatible types, precision, and scoping.
- Importance of error handling with try, catch, and finally blocks.
- Checked versus runtime exceptions.

14.1 | Introduction

Error handling is an important concept in all programming languages. It must be learned and implemented with accuracy to ensure that problems do not ensue in the long run. Even though most people are intimidated by error handling and believe it to be daunting and complicated process, the fact remains that this process is imperative for efficient and effective programming.

Tossed around often, the term “error handling” essentially refers to an entire world of anticipating the possibility of code errors, creating mechanisms to detect them and ultimately, resolving these errors and anomalies in code using various programming and communicating processes. To help put things into perspective, we will first describe error handling from a programmer’s point of view to understand how significant error handling is actually for creating and implementing a well-structured program or piece of code.

Next, we will shed some light on the different terms that are involved in error handling and exception handling situations including try, catch, and finally. Then, we will differentiate between runtime and compilation errors. We will also explain how and why the exceptions and errors in both cases vary so greatly, especially in Java programs.

Towards the end of our discussion, we will also shed some light on the different techniques that can be used for error and exception handling. We will also explain how each type of error or exception should be handled based on the situation. This will help programmers get a better understanding of how problems, inaccuracies, and anomalies in the code or program should be dealt with in a manner that solves the problem effectively and efficiently as possible.

14.2 | Understanding Error Handling

Error handling involves a lot more than just the removal of errors. In fact, error handling refers to the entire system of procedures, techniques, and processes that are required to anticipate and identify where problems lie before we get to deal with an error or an exception in the code itself.

Anticipating and knowing that there is a perpetual possibility of errors or exceptions manifesting themselves at any place in your code is the first step involved in effective error and exception handling. This is because once you begin to write code keeping this fact in mind, you will naturally begin to incorporate best practices in your code, minimizing the possibility of errors and exceptions altogether. When errors are anticipated and the possibility of their manifestation is kept in mind, chances are that you will also double check your code more often than you otherwise would in order to minimize problems in the future.



Since errors and exceptions can even occur despite checking the code multiple times, it is imperative for programming languages such as Java to have an error handling system in place, allowing developers to implement a systematic approach to resolve them.

Unlike most concepts that deal with either the software or hardware side exclusively, error handling is one of the few concepts that erases the distinguishing line between the two. Error and exception handling does not only help identify both software and hardware problems, but also allows programmers and developers to deal with the problem at hand in an effective manner, ensuring that the functionality of the rest of the program is not compromised. Needless to say, without the right error and exception handling procedures and techniques in check, it would be impossible for programmers and developers to change one block of code without affecting the rest of it in the process.

Fortunately, in today's day and age, programmers have two major options when it comes to error and exception handling. Programmers can either develop codes and programs that allow room for dealing with errors, or they can make use of software and tools available online and elsewhere to handle problems that are caused by errors and exceptions. While the distinction of the type of error is often clear, there are certain cases in which an error may seem to fall either in a number of different categories, or it may be difficult to identify a category at all due to the ambiguous nature of the error or exception. In cases like these, it is suggested that you make use of customized software for the identification of the errors and exceptions.

There are four major categories of all errors in Java and a majority of other programming languages. These categories are logical errors, generated errors, compile-time errors, and run-time errors. Needless to say, there are different techniques and processes that are involved in dealing with errors of each of these different categories. While the errors in certain categories may be solved and avoided altogether with the help of some basic proofreading of the code, other types of errors may require the programmer or developer to identify the problem with the help of test data and deal with it using resolution programs.

Figure 14.1 illustrates the exception hierarchy in Java.

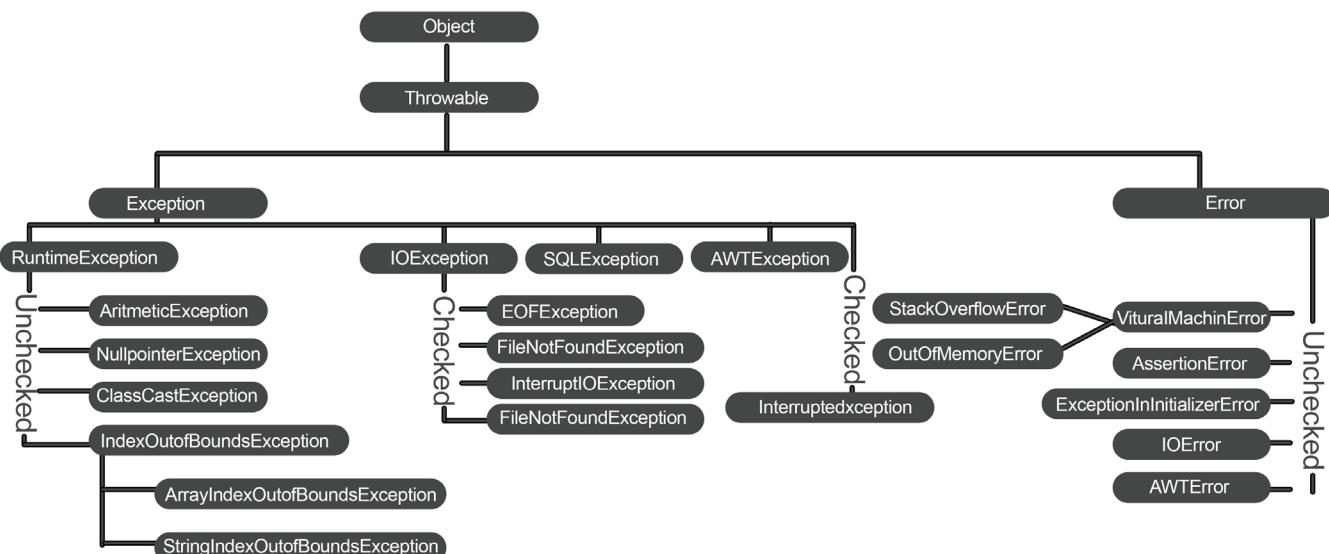


Figure 14.1 Exception hierarchy in Java.

QUICK CHALLENGE

Create a chart which shows the difference between Exception and Error. Also provide one example of each.

14.3 | Logical Errors

Logical errors are often considered to be the most difficult types of errors to spot. This is primarily because instead of causing a program to terminate or stop working altogether, logical errors and exceptions produce incorrect results. What this means is that a program with logical errors will run perfectly fine, but you will not be able to see the results which you anticipated, be it due to a typo or incorrect usage in terms of logical operator precedence.

To help you understand better the importance of keeping logical operator precedence in mind, we will illustrate the damage that may arise with the help of the example below. You can see how something as simple as the different usage of parentheses can produce different outputs of 11, 13, 9, and 8.



```

package java11.fundamentals.chapter14;
public class LogicalErrors {
    public static void main(String[] args)
    {
        // Create variables Var1 thru Var4
        int Var1 = 5 + 4 * 3 / 2;
        int Var2 = (5 + 4) * 3 / 2;
        int Var3 = (5 + 4) * (3 / 2);
        int Var4 = (5 + (4 * 3)) / 2;
        // Print the results.
        System.out.println(
            "Var1: " + Var1 +
            "\nVar2: " + Var2 +
            "\nVar3: " + Var3 +
            "\nVar4: " + Var4);
    }
}

```

The above program produces the following result.

Var1: 11
nVar2: 13
nVar3: 9
nVar4: 8

Since logical errors have more to do with the logic of a program than the actual structure, simple proofreading often suffices for identification and resolution of such errors. While proofreading your code is a practice that should always be implemented, it is essential to double check your code several times especially if it involves a lot of logic building. This is also because logical errors can cause situations and problems that are a lot more severe. In some cases, a condition that is false may even be assumed as true due to incorrect operator usage – something that has the potential to cause errors and problems that will carry on in the rest of the program.

In certain cases, logical errors that have not been spotted at an early stage in the code or program may even have the potential to affect data and values that appear thousands of lines of code after the error was initially made.

One of the most common logical errors that bother programmers is misuse or misplacing of a semicolon. While misplaced semicolons may not give you the results that you had intended, the fact remains that it is completely possible to create a fully functional Java code with misplaced semicolons.

Here is an example of how misplaced semicolons work:

```

package java11.fundamentals.chapter14;
public class ErrorForLoop {
    public static void main(String[] args)
    {
        // Variable Declaration.
        int Counter;
        // Create For Loop.
        for (Counter = 1; Counter <= 10; Counter++)
        ;
        {
            // Print the result.
            System.out.println("Counter is " + Counter);
        }
    }
}

```

The above program produces the following result.

Counter is 11

In the above example, the only output that will be displayed to the user will be “Counter is 11”. This is because the semicolon was placed right after the creation of the *for* loop, instead of placing it at the end of the code block. Had the semicolon been placed at the end of the entire *for* loop block, the output would be a list of individual values for Counter starting from 1 and finishing at 11.

As already mentioned, most logical errors can be removed by proofreading, where you go through each line of code and make sure that all the coding elements are properly set up to avoid such errors. In fact, it is also possible to avoid them in most integrated development environments (IDEs) that provide color recognition of different program elements. Programmers can quickly find out where they may have left a code situation that will result in a logical error.

Techniques that are employed for error handling are often termed as *debugging* or *troubleshooting*. Runtime errors are also quite significant in regard to *debugging* or *troubleshooting*, and they are often handled by setting up countermeasures in the programming environment to avoid such situations altogether. In fact, applications that run on hardware are often designed to have an error handling method, which allows the application to recover from any error and once again restart to provide functionality to the program.

14.4 | Syntactical Errors



Syntactical errors are errors in which the wrong language, or syntax, is used to write a code or program. Not writing the condition in parentheses for an *if* loop, for instance, would classify as a syntax error since that is a requirement for the code to run properly. This even holds true if the condition were to be present on the same line as the *if* statement, but just not in parentheses.

Syntactical errors may not be as difficult to spot as their logical counterparts because the compiler will most likely catch the majority of these errors for you. However, as always, proofreading and knowing the correct syntax and conditions for all loops and codes is essential to ensure that you do not have to face any problems in the long run.

The reason why it is so important to spot and resolve syntactical errors as early as possible is because if a code or program has syntactical errors, it will not be possible for the Java Runtime Environment (JRE) to use the byte code that needs to be created by the compiler. Since syntactical errors have the potential to cause a lot of problems, we are sharing some of the most common syntactical errors and why they are so important to solve.

QUICK CHALLENGE

Write a program that can demonstrate syntactical errors.

14.4.1 Capitalization

Java is a case-sensitive language but not a lot of new programmers realize that. This is the major reason why so many new Java programmers start capitalizing keywords instead of writing them in lowercase. Capitalization may or may not make a difference in certain languages; however, in languages such as Java, making a change as apparently insignificant as writing myCount instead of MyCount has the potential to ruin the entire code and fill it with more errors than you ever thought would be possible.

Using the right capitalization is essential for all class names, variable names, and any other piece of code that you will be writing in Java.

14.4.2 Splitting Strings

Dividing your code into a number of lines often does not matter when you are coding in Java. However, there is an exception that applies when you are adding strings in your program or code. Splitting a string so that it comes on more than one line or contains a new line character in it will cause the compiler to throw an exception or object to the code that you have written.

Fortunately, there is a way to create strings that absolutely have to be written in multiple lines. The approach that is required to do this without receiving any errors messages, exceptions, or objections is to add a double quote to the string that appears on

the first line, and then add a plus sign right after this first half of the string ends to show the compiler that whatever follows in the next one or more lines needs to be added or concatenated to the same string. An example of how this can be done is as follows:

```
System.out.print ("This is the first half of the string " +
"this is the second half of the string that needs to be concatenated. " );
```

14.4.3 Not Importing Classes

One of the most common semantic errors that programmers – both new and seasoned – make when coding in Java is forgetting to include an associated class when they wish to make use of a particular API feature. For instance, if you wish to incorporate the String data type in your code, it is essential to add the class to your application using `Import java.lang.String;` for the String class to be imported in your application.

14.4.4 Different Methods

When coding in Java, you must remember that static methods and instance methods work differently in this language. Static methods are those that are associated with a specific class, whereas instance methods are associated with the object that is created from a certain class. In case you treat a static method as an instance method in Java, your compiler will present you with a syntactical error since the way in which both of these are dealt with differ greatly.

14.4.5 Curly Braces

When programming in Java, you will often find yourself in situations where you want the same feature to apply to more than one-line code. In cases like these, it is imperative for you to create a block of code enclosed in curly braces to ensure that the compiler understands where the code that the feature needs to be applied to starts and finishes. While the compiler may catch this error for you in most cases, it is still important for you to keep an eye out for lines and blocks of code that need to be treated as a single entity and make sure that they are distinctly identifiable by the compiler.

For instance, if you forget to finish the contents of a class with a curly braces, this will be treated as a syntactical error by the compiler and you will be notified of a missing curly braces. In the example below, the class Cat will not be recognized because the compiler will not know where it ends.

```
package java11.fundamentals.chapter14;
public class Cat {
    int age;
    String breed;
    String color;
    void meowing() { }
    void sleeping() { }
    void hungry() { }
}
```

As seen in the above example, each of the methods – namely meowing, sleeping, and hungry – do not have any contents in them but they still have curly braces to show where they start and end. The class Cat, on the other hand, does not end with a curly braces which is why an error or exception will be generated.

Moreover, in the scenario mentioned above, while the compiler knows that a curly braces is missing, it may not be able to pinpoint the exact location where the curly braces should appear. This is because each of the methods may or may not have been a part of the Cat class. This is why you will only be notified that a curly braces is missing but not where it should appear.

Since using a curly braces in Java is the syntactical rule when a feature or action needs to be applied to multiple lines of code, runtime errors can also occur in case you forget to add a curly braces or you add one in the wrong spot. In case you forget to add a curly braces at the end of an `if` statement, the condition will only apply to the line of code that comes immediately after the `if` statement and can potentially cause problems in the way the application or program was intended to run.



14.5 | Semantic Errors

Figuring out the difference between semantic and syntactical errors is one of the biggest problems for both beginners and seasoned programmers of the Java language. While the majority of people tend to classify both semantic errors and syntactical errors in the same category due to ease and convenience, there are certain significant differences between these types of errors. While syntactical errors have to do with the syntax of the code, semantic errors are related to the usage of the code. This means that it is possible for you to have semantic errors in your code even if the syntax is correct.

While you might have probably already guessed this, the most common type of semantic errors are those in which variables are used without proper initialization. You already know that a variable cannot be used or be expected to add value to your code in case there are problems with its declaration or its initialization. Fortunately, these errors will be caught by the compiler in most cases and you will receive a notification about the variable in question.

While the problem with variables is the most common semantic error, there are plenty of others that you should be aware of as a programmer of the Java language. Subsequent subsections discuss some of the most common semantic errors and explain why each of them occurs. In some cases, we may also tell you how they can be solved or avoided altogether.



What is the difference between syntactical errors and semantic errors?

14.5.1 Improper Use of Operators

Sometimes, in case of operators are used improperly on variables, it may give an impression of syntactical error. However, in reality it is more of a semantic error. For example, the increment operator (++)⁴, for instance, cannot use Boolean variables and attempting to do so will be a semantic error.

While new versions of Java are able to detect these problems far more easily, finding out exactly why an error message is generated may be a bit of a challenge especially if you are not sure what operators are allowed to be used with what variables.

Another common operator error mistake the use of comparator (==) operator with objects. Since this is not allowed and comparator operators can only be used with primitive types, an error message will be generated showing that the operation intended is not permissible.

14.5.2 Incompatible Types

Whether it is done by accident or simply due to the lack of knowledge, programmers of the Java language often try to use incompatible types together. Needless to say, doing so is a semantic error that may or may not be caught by the compiler.

For instance, if you mistakenly try to assign a float value to an int variable, the compiler will present an error message. However, if you try to assign an int to a float, the compiler will automatically convert the integer value into a float value. This could potentially cause several problems, especially if that conversion was never intended. Additionally, since the programmer or user will not be notified of this conversion as the compiler will practically expect that this was the desired output, it will become almost impossible for the anyone working on the code to find out that the code contains an error.

14.5.3 Precision

While a float variable can be converted to an int variable by applying casting, doing so incorrectly can result in a loss of precision. Additionally, since everything after the decimal will automatically be lost, the precision of the value that you use will be affected along with the results in all of the other lines or blocks of code where the value in question will be used. It is, therefore, recommended that you only use casting when you absolutely must and know that your output can potentially be affected.

14.5.4 Scoping

Scoping is an issue that tends to bother even the most experienced programmers. This is because there are quite a number of rules that define what is and is not allowed within a certain scope. For instance, if you try to declare a private static int variable inside a method, you will receive an error. Instead, you should declare the variable globally so that it can be used properly. Following programs are examples of incorrect and correct ways to declare a private static int variable.

If you have a class called *VariablePrivate* and you declare the private static int variable globally in the class itself, it will work as follows:

```
package java11.fundamentals.chapter14;
public class VariablePrivate {
    // This is the correct way to declare the private static int variable called
    intPrivate.
    private static int intPrivate = 5;
    public static void main(String[] args)
    {
        // The contents of the method will go here
        System.out.println("intPrivate value : " + intPrivate);
    }
}
```

The above program produces the following result.

intPrivate value : 5

On the other hand, if you try to declare the private static int variable called *intPrivate* within the method itself, you will get an error message. Following is the incorrect way of declaring the private static int variable:

```
package java11.fundamentals.chapter14;
public class VariablePrivateIncorrect {
    public static void main(String[] args)
    {
        private static int intPrivate = 5;
    }
}
```

The above program produces the following result.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  Illegal modifier for parameter intPrivate; only final is permitted
  at java9.fundamentals.chapter6.VariablePrivateIncorrect.main(VariablePrivateIncorrect.java:10)
```

14.6 | Importance of Error Handling

The importance of error handling can never be emphasized enough. Error handling does not only ensure smooth operation but also guarantees that the code will not malfunction and will provide the desired results.

Since even the simplest of applications are easily a few thousand lines of code long and comprise code written by a number of programmers, it is extremely important to take care of all errors and exceptions as you move forward. This will ensure that errors, exceptions, and other problems of the sort do not carry forward until the end of the program.

14.6.1 Try, Catch, and Finally

Earlier, we talked all about errors and their different types along with ways in which they can be avoided or resolved. By now, you probably already understand the importance of error handling and realize that it is imperative to resolve errors as soon as they are spotted to ensure that they do not continue to cause problems in the rest of the code. While error handling is something that you are probably already aware of, there is another thing that you should probably worry about – exceptions.

So, what exactly are exceptions and how do they work? As the name suggests, exceptions derange the regular flow of the program or code and make it act in a way that it should not. While there is nothing wrong with the syntax of the code in which an exception is being caused, exceptions still cause problems and make your code or application act in a way that is different than what was expected.

In Java, the concept of exceptions is a lot more profound. Error events in Java are wrapped by exceptions that occur within a method. Exceptions in Java do not only contain information about the error that occurred and the type of the error that has manifested itself in the code, but also details the state of the program when the error occurred. Additionally, some other custom details that can help you assess the nature of the error and the impact that it caused on the code may also be included in the exception.

Fortunately for Java programmers, exceptions can point out a multitude of different error conditions that may occur within the code. When talking about Java virtual machine (JVM) errors, exceptions cannot only help indicate OutOfMemory errors and StackOverflow errors, but they can also help the programmer understand Linkage errors and why they occurred within the code along with details about the error itself. Exceptions can also help programmers understand System errors, including FileNotFoundException exceptions, IOExceptions, and SocketTimeOutExceptions.

The reason why many programmers of Java language are interested in using exceptions is primarily because these allow them to treat the regular flow of a code as a separate entity, unlike error handling. As a result, you do not only get clearer algorithms that are far easier to handle and understand than regular code, but the clutter within the code also decreases significantly, allowing you to be more creative and innovative with your programs and applications.



Can you delay the exception handling further down the method calls?

In programs or pieces of code where an exception is possible, it is recommended that you use a statement that will be able to catch the exception. By doing so, you can prevent the entire program from crashing should the exception occur. One of the most common statements used for this purpose is the “try” statement. By incorporating a try statement in your code, you can be sure that the potential exception will be caught and treated as a block of code that is separate from the remaining program, code, or application that it is a part of. The general form of the try statement is as follows:

```
try
{
// statements that can potentially cause an exception will go here
}
catch (identifier of type of exception)
{
//statements that should be executed if the exception is thrown will go here
}
```

As seen above, the try statement helps you treat a block of code with the potential of an exception as a separate entity to ensure that the rest of the program is not affected. Additionally, it is seen that the try statement also accommodates an alternative statement or block of code that should be executed should the exception be thrown, as seen in the second half of the example above.



Can you capture errors using try-catch?

In try-catch statements, it is possible for you to code multiple try blocks. This particularly comes in handy where the statements in the try block throw exceptions of different types. Additionally, from Java 7 onwards, it is even possible to catch multiple types of exceptions within the same catch block by separating the type using vertical bars. An example of how this is done is as follows:

```
try
{
    // statements that have the potential to throw
    // ClassNotFoundException
    // ArrayIndexOutOfBoundsException
}
catch (ArrayIndexOutOfBoundsException | ClassNotFoundException e)
{
    System.out.println(e.getMessage());
}
```

It is also important to note that the contents of the try block are not visible to the catch block, which is why it is impossible for any variables declared in the try block to be used in the catch block or blocks. In case there is a variable that you need to use in both blocks of code, it should be declared before the try block.

The following is another example of how the try-catch statement can be used in action to prevent the user from trying to divide the value of a variable by 0.

```
package java11.fundamentals.chapter14;
public class DivideByZero {
    public static void main(String[] args)
    {
        int var1 = 5;
        int var2 = 0; // 0 is assigned to var2 to cause an exception by dividing var1 by 0
        try
        {
            int var3 = var1 / var2; // This is the statement that will cause the exception
        to be thrown
        }
        catch (ArithmaticException e)
        {
            System.out.println("It is not possible to divide by zero");
        }
    }
}
```

The above program produces the following result.

It is not possible to divide by zero

Another block of code that is used for exceptions in Java is the finally block. One unique and interesting feature about the finally block is that it is always executed regardless of whether or not any exceptions have been thrown in the code. With that said, using the finally statement is among best practices and is expected to be used particularly in scenarios when you are closing a connection or file. Here is an example of a program where the finally block will be executed:

```

package java11.fundamentals.chapter14;
public class FinallyBlockExample {
    public static void main(String args[])
    {
        try
        {
            int var = 30 / 6;
            System.out.println(var);
        }
        catch (NullPointerException e)
        {
            System.out.println(e);
        }
        finally
        {
            System.out.println("These are the contents of the finally block");
        }
        System.out.println("The finally block has been executed");
    }
}

```

The above program produces the following result.

5
These are the contents of the finally block
The finally block has been executed

Since no exception will be thrown, the value of the variable “var” will be displayed to the user, after which the finally block will be executed and the words “These are the contents of the finally block” will be displayed. Next, the words “The finally block has been executed” will be displayed to the user.

14.7 | Checked verses Runtime Exceptions

As the name suggests, checked exceptions are those that are identified at the time when the code is being compiled. On the other hand, runtime exceptions are identified when the code is being run.



Can program execution continue even after an exception is thrown?

14.7.1 Checked Exceptions

In case a checked exception is being thrown by a method, it is necessary that this exception should either be handled by the method itself, or the *throws* keyword should be used to specify it. The following example shows how a checked exception is thrown in the *main()* function.

```

package java11.fundamentals.chapter14;
import java.io.*;
public class CheckedExceptionsExample {
    public static void main(String[] args)
    {
        FileReader file = new FileReader("D:\\\\newfolder\\\\example.txt");
        BufferedReader fileInput = new BufferedReader(file);
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());
        // This block of code will output the first 3 lines of the file
        // "D:\\newfolder\\example.txt"
        fileInput.close();
    }
}

```

The above program produces the following result.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
  Unhandled exception type FileNotFoundException
  Unhandled exception type IOException
  Unhandled exception type IOException

  at java9.fundamentals.chapter6.CheckedExceptionsExample.main(CheckedExceptionsExample.java:9)
```

In the example above, the `main()` function uses `FileReader` to read the file located at `D:\\newfolder\\example.txt`. (If you are executing this code on your end, make sure you change the file location as per the location of file on your computer). However, a checked `FileNotFoundException` is thrown by `FileReader()`, whereas checked `IOException` is thrown by the `close()` and `readLine()` methods. As a result, a message will be displayed showing where the exception was thrown, explicitly stating that the source code cannot be compiled.

14.7.2 Runtime Exceptions

Runtime exceptions in Java are a subcategory of unchecked exceptions or exceptions that are not checked during the compilation of the code. Since unchecked exceptions and runtime exceptions are not detected at the time of compilation, the code compiles perfectly before the runtime exception is detected when the program is run. The following is an example of a runtime exception:

```
package java11.fundamentals.chapter14;
public class RuntimeExceptionExample {
    public static void main(String args[])
    {
        int var1 = 0;
        int var2 = 10;
        int var3 = var2 / var1;
    }
}
```

The above program produces the following result.

```
Exception in thread "main" java.lang.ArithmaticException: / by zero
  at java9.fundamentals.chapter6.RuntimeExceptionExample.main(RuntimeExceptionExample.java:9)
```

In the example above, the `main()` function will throw an `ArithmaticException` which is a type of runtime exception, as it is not possible to divide by 0. However, since there is nothing wrong with the syntax of the code, it will compile perfectly before the exception is detected when the program is run.

QUICK CHALLENGE

Write a program which can demonstrate all types of Runtime Exceptions.

Summary

In this chapter, we have discussed error handling and how to use it. We also discussed the concept of error handling and elucidated various types of errors such as logical, syntactical, and semantic. Then we discussed the importance of error handling in which we studied the use of try, catch, and finally blocks. At the end of the chapter, we learnt the difference between checked and runtime exception with examples.

In this chapter, we have learned the following concepts:

1. What is error handling? What is the importance of using it?
2. What are logical errors, syntactical errors such as capitalization, string split, missing or improper imports of classes, and missing curly braces?
3. What are semantic errors such as improper use of operators, incompatible types, precision, and scoping?
4. How do we handle errors using try, catch, and finally blocks?
5. What is the difference between checked and runtime exceptions?

In Chapter 15, we will learn about garbage collection, using it in a program, and how to implement the garbage collector.

Multiple-Choice Questions

1. Exceptions arises during _____ in the code sequence.
 - (a) Compilation time
 - (b) Run time
 - (c) Can occur anytime
 - (d) None of the above
2. _____ is not an exception handling keyword.
 - (a) finally
 - (b) thrown
 - (c) catch
 - (d) try
3. Exception can be thrown manually by using _____ keyword.
 - (a) finally
4. Which of the following is the parent of Error?
 - (a) Object
 - (b) Collections
 - (c) Throwable
 - (d) Exception
5. What do you understand by unchecked exceptions?
 - (a) Checked by Java virtual machine
 - (b) Checked by Java compiler
 - (c) (a) and (b)
 - (d) None of the above

Review Questions

1. What is error handling?
2. How is error handling useful?
3. What is the difference between error and exception?
4. Which exception is thrown when no class is found?
5. How do try, catch, finally blocks work?
6. What are semantics errors?
7. What are logical errors?

Exercises

1. Write a program that can produce exceptions and catch them using try, catch, finally blocks.
2. Create a comparison chart to distinguish between error and exception.
3. Write a program that can produce errors. Observe and document the outcome.

Project Idea

Create a calculator program that performs various types of arithmetic operations. Also create a program that can divide any number by any number. Make sure you add exception

handling to capture and notify users in case they use unpermitted operations, such as dividing a number by 0. Make sure your program has all the features of a normal calculator.

Recommended Readings

1. Oracle Tutorials – https://www.w3schools.com/java/java_tryCatch.asp
2. W3Schools – <https://docs.oracle.com/javase/tutorial/essential/exceptions/>
3. Oracle Technetwork – <https://www.oracle.com/technetwork/java/effective-exceptions-092345.html>

Garbage Collection

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Java memory management.
- Garbage collection.
- How to code according to the garbage collector.
- How to make objects eligible for collection.
- The latest updates of garbage collection.

15.1 | Introduction

Memory management is always an extremely important part of any program or application being developed in the Java programming language. Whenever objects are created for classes in any programs or applications developed using Java language, they are allocated a certain amount of memory, allowing the program or application to function the way that it should.

Just like objects in Java, variables are also allocated memory that allows them to be stored and used throughout the program. All variables and objects need to be assigned or allocated certain areas of memory to ensure that they can be used without errors or exceptions manifesting themselves. However, the memory that is allocated to them differs according to the scope that the said variable is located in.

Once the memory allocated to a certain variable or object is deemed useless and the purpose of the variable or object has been completed, it is important for the allocated memory to be recycled and be usable for other purposes. And that is where the garbage collector in Java comes into play.

Fortunately for developers of the Java programming language, memory reclamation is automatic in the Java virtual machine (JVM), which means that Java developers do not necessarily have to go out of their way in order to free memory objects that are no longer being used. Garbage collection in Java also works based on the assumption that objects are short-lived and can be easily reclaimed once they have been created.

Another plus in Java is that if there are ever any objects that are not referenced, they are automatically removed from heap memory to free up space for other objects and variables, making this an extremely memory-efficient language.



What will happen to your program at runtime if there is no Garbage Collection taking place?

15.2 | Garbage Collection in Java

Since garbage collection has to do with memory management more than anything else, it is imperative for us to talk about how memory works in Java programming. It is clear that garbage collection in Java is an automatic process, and that the programmer or developer creating a piece of code or developing an application does not need to go out of their way in order to tell the machine which objects in the code need to be deleted. However, there are certain specifications that need to be met to ensure that the object will be deleted. And this is where memory management is important.

For the sake of simplification, let us say that the memory heap for Java is divided into three major sections – Young Generation, Old (or Tenured) Generation, and Permanent Generation.



Whenever a new object is created, it is located in the Young Generation. This Young Generation is further divided into two subcategories – Eden Space and Survivor Space. Upon creation, new objects are present in the Eden Space and are moved to Survivor Space 1 after the first garbage collection cycle.

A minor garbage collection event then follows in order to move objects from the Young Generation to the Old Generation. The Old Generation contains all objects that have matured enough to be moved from here but cannot fall in the category of Permanent Generation. When the garbage collector needs to remove objects from the Old Generation, this is known as major garbage collection event.

All data that is required for proper functioning of the application or code is stored in the Permanent Generation. This generation, therefore, stores meta data and information regarding classes. In case there is a class that does not need to be used, it can be removed from the Permanent Generation with the help of a garbage collector in order to free memory for other classes or data that is imperative for the code to run properly.

Most developers regard the Permanent Generation as a block that is contained in the native memory instead of the heap memory. Since the Permanent Generation contains class definitions by class loaders, this block has inherently been designed to expand and grow to ensure that there are no out of memory errors or exceptions that are thrown in the code. However, in case the block needs more memory than available in the physical memory, the operating system ensures that virtual memory is made available for the code to run like it should. This virtual memory will certainly allow the code to run; however, in order to make use of this virtual memory, the constant back and forth will be required between the virtual memory and physical memory. This affects the performance and smoothness of the code.

Now that you have a basic understanding of memory heap and how it works, we can start talking about the process that is involved in garbage collection. A daemon thread is created and used by the JVM for garbage collection. Whenever a new object is created, the JVM attempts to get the space that is required for the object from the Eden Space. As is the rule, the Survivor Spaces and Tenured Space are empty at the beginning of the code.

In case the JVM is unable to find the required memory from Eden Space, minor garbage collection is initiated to free up the required space. For this process, one of the two Survivor Spaces, S0 or S1, are regarded as the To Space. Next, all objects that are not reachable are copied by the JVM to the To Space, and 1 is added to their age. On the other hand, all objects that are not fit for the Survivor Space are moved to Tenure Space.

Since not every object is meant to move from the Young Generation Space to the Tenured Space, JVM comes with a Max Tenuring Threshold. This is basically an option that can be modified according to the preferences of the programmer or the requirements of the application to ensure that there is always enough memory for the creation and initiation of new objects. While the default value of the Max Tenuring Threshold is set as 15, it can be changed.

As mentioned earlier, a minor garbage collection process occurs in order to reclaim memory that can be freed from the Young Generation (when objects become mature and move on to Tenured Space). It is important to note that garbage collection is a Stop The World process in Java, which means that the garbage collector ensures that all the threads that are being used to run the application or program are stopped and only the threads that are being used for garbage collection are still running until the process is complete. It is also important to keep in mind that Stop The World will occur regardless of the algorithm that is being used for garbage collection.

The number of threads being used for garbage collection will depend on the algorithm that is being used for the process. Based on the algorithm, garbage collection could either be done successfully using a single thread or multiple different threads working together to clean out memory. Additionally, while the delay caused by the STOP-THE-WORLD application is often negligible, in cases where there is a lot of memory to be cleaned, garbage collector tuning can also be applied to reduce the STOP-THE-WORLD time.



Can we guarantee garbage collection?

15.3 | Major Garbage Collection

If minor garbage collection occurs very frequently, the objects from the Young Generation will naturally move into Tenured Space and occupy all of the available memory very quickly. Since that will prove to be detrimental to the program or application, JVM will trigger a major garbage collection event. While major garbage collection is also referred to as full garbage collection at times, it should be noted that full garbage collection entails reclaiming memory from the Meta Space as well.

And while this is one way in which major garbage collection can be triggered, there are a number of other possibilities as a result of which JVM can call major garbage collection. Even though it is generally advised against, if a programmer decides to

call `Runtime.getRuntime().gc()` or `System.gc()`, the JVM will trigger a major garbage collection. It is also possible for major garbage collection to be triggered if there is not enough memory remaining in the Tenured Space, if the JVM is unable to reclaim the required amount of memory from the Eden Spaces or Survivor Spaces, or if enough space is not available for the JVM to load new classes or objects as they are created in the program or application.



Is there any situation where the garbage collector stops working?



15.4 | G1 and CMS Garbage Collectors

Java offers various different types of garbage collectors, which have their own advantages and disadvantages. It is important to learn about garbage collectors to understand which one to use for your specific needs. The following are the two most significant garbage collector options offered by Java:

1. **Garbage First (G1) garbage collector:** Introduced in Java 7, G1 is capable to handle very large heaps efficiently and concurrently. For Java 9, this is the default garbage collector. If you are using a version prior to Java 9, you may enable G1 with `-XX:+UseG1GC` parameter for JVM. G1 offers various advantages:
 - (a) Uncommitting unused heap.
 - (b) Free up memory space without using a long pause time.
 - (c) Work concurrently such as without interrupting or stopping application threads.
 - (d) Deal with very large heap by using non-continuous spaces.
 - (e) It can collect both young and old generation spaces at once. This can be achieved by G1 by splitting the heap into hundreds of small regions instead of only three (i.e., Eden, Survivor, and Old) like most other garbage collectors do.

Mainly, GC1 outshines other garbage collectors on large amount of data as it does not have to work on the entire heap or entire generation. It can simply work on the selected small regions and finish quickly. On the disadvantage side, GC1 struggles to work with small heaps.

2. **Concurrent Mark Sweep (CMS) garbage collector:** As the name suggests, Concurrent Mark Sweep uses multiple threads (“Concurrent”), which are used to scan through the heap and mark the unused objects (“Mark”) that can be collected and recycled (“Sweep”). Many applications strive for shorter garbage collection pauses and do not get affected by sharing their processor resources with the garbage collector while the application is running. These are the perfect candidates to use CMS. Also, the benefits of this garbage collector are for the applications which got a large set of long-lived data (such as a large tenured generation) and execute on multiprocessors. You can enable the CMS collector with the following command-line option:

`-XX:+UseConcMarkSweepGC`

There are a few challenges in using CMS collector, such as finding the right time to initiate the concurrent work, as this work can get completed before the application is out of available heap space. CMS requires higher percentage of heap space than Parallel garbage collector in a scale of 10% to 20%. Hence, it is a costlier proposition for using shorter garbage collector pause times. Another challenge is related to handling the fragmentation in the old generation. When old generation goes through the garbage collector process, it may occur that the free space between objects get smaller or nearly non-existent. Hence, the objects which are getting promoted from the young generation do not find sufficient place to fit. Since CMS concurrent collection does not do any type of compaction, whether incremental or partial. This unavailability of space for promoted objects forces CMS to a full collection using Serial garbage collector. This results in a lengthy pause.

QUICK CHALLENGE

Write a memory-intensive program which creates a lot of objects. Try G1 and CMS collector on this program. Print timestamp and heap size. Use the following commands to print the heap size and free space.

Command to print total memory of heap:

```
Runtime.getRuntime().totalMemory()
```

Command to print free memory of heap:

```
Runtime.getRuntime().freeMemory();
```

15.5 | Advantages of Garbage Collection in Java



Garbage collection in Java is essential for its benefits pertaining to freeing up memory to ensure that it can be used for other purposes. However, limiting the advantages of garbage collection to just that is an injustice of the highest degree. Since you are not responsible for keeping tabs on the data and figuring out when it is no longer necessary for a certain object or needs to be cleaned after being left behind by an object that is not referenced anymore, you do not only have the time to deal with everything else that is on your plate, but automation of the garbage collection process also makes you more productive.

And that is not all. Since manual garbage collection in Java has been made extremely difficult, it is possible for programmers or developers to accidentally cause the program or application that they are working on to crash due to incorrect removal of certain objects from memory. Additionally, since removal and updating of memory works automatically in the programming language, integrity of the program is maintained at all times.

15.6 | Making Objects Eligible for Garbage Collection



The only reason why an object will be suitable for garbage collection in Java is if the reference variable of the object is no longer available. Objects that fall under this category are also termed as *unreachable objects*.

While there are a number of different ways in which an object can be made eligible for garbage collection, it is important to first talk about the reference of an object and how referencing works. By now, you are probably aware that every object is assigned a certain amount of memory from the available heap, but the operation behind the process is slightly complicated. Every time an object is automatically assigned memory through the operator, a reference to that memory is returned back. For ease of understanding, this reference is basically the address of the newly created object in the memory that is given to it by the “new” operator.

At this point, it is important to note that if an object is being referenced, it is not necessary that it is being used at that particular moment in the program or application. What this means is that if an object is passed as an argument or assigned to a variable, this action essentially only takes the reference of the object into account and does nothing more. An example of this type of referencing is as follows:

```
Book myJavaBook = new Book();
```

15.6.1 Unreachable Objects

As mentioned earlier, only objects that no longer have a reference associated with them can be deemed or made eligible for garbage collection. Why is that the case? This is because an object that does not have a reference is essentially not present on the memory heap, which means that it is not available for use and cannot add any value. Figure 15.1 shows the unreachable and reachable objects.

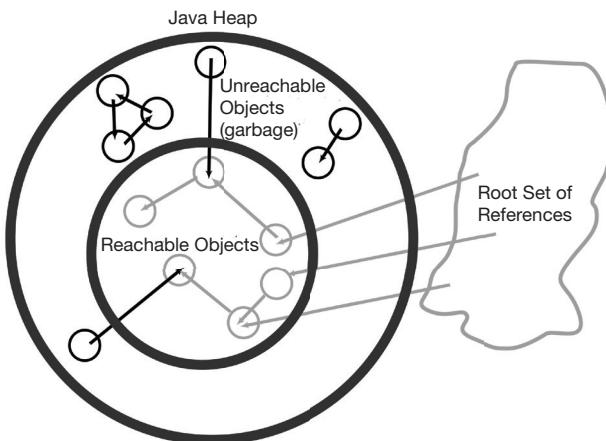


Figure 15.1 Reachable and unreachable objects in Java heap.

Since we are talking about how objects can be made eligible for garbage collection, the first scenario where this process can be applied is when objects are created within the scope of a method. Whenever a method is called, it is pushed on or moves directly to the stack that contains all methods that are important for the successful execution of the program or application. Now, when this method is popped or removed from the stack, all of the members that were associated with this method die with it.

In case there were any objects that were created in this method, they will also die off, leaving unreferenced objects on the heap that can no longer be used for any sort of value addition. Based on the premise that they do not have any reference, these anonymous objects automatically become eligible for garbage collection.

Here is a program to demonstrate that objects that are created within the scope of a method will be deemed useless after execution of that method is complete.

```
package java11.fundamentals.chapter15;
public class UnreachableObjectsExample {
    private String myObject;
    public static void main(String args[])
    {
        // Executing testMethod1 method
        testMethod1();
        // Requesting garbage collection
        System.gc();
    }
    public UnreachableObjectsExample(String myObject)
    {
        this.myObject = myObject;
    }
    private static void testMethod1()
    {
        // After existing testMethod1(), the object myObjectTest1 becomes unreachable
        UnreachableObjectsExample myObjectTest1 = new
        UnreachableObjectsExample("myObjectTest1");
        testMethod2();
    }
    private static void testMethod2()
    {
        // After existing testMethod2(), the object myObjectTest2 becomes unreachable
        UnreachableObjectsExample myObjectTest2 = new
        UnreachableObjectsExample("myObjectTest2");
    }
    @Override
    protected void finalize() throws Throwable
    {
        // following line will confirm the garbage collected method name
        System.out.println("Garbage collection is successful for " + this.myObject);
    }
}
```

Since both the objects within the method had become unreachable, the output will be as follows.

Garbage collection is successful for myObjectTest2
Garbage collection is successful for myObjectTest1

The output shows that any object within a method becomes useless after execution of the method; the object automatically becomes eligible for garbage collection.

15.6.2 Reassigning Reference Variables

Reference IDs are extremely important in Java and help in addressing each of the objects and variables that are being used in any code. In case, one object's reference ID is used to refer to other object's reference ID, then the first object that was initially being referenced becomes unreachable and cannot be used in the program or code in any way. Once it becomes unreachable due to the reference ID being used for multiple objects, the first object is deemed eligible for garbage collection.

The following program is an example of a situation where a reference ID is used to reference multiple objects.

```
package java11.fundamentals.chapter15;
public class ReassigningReferenceExample {
    private String myObject;
    public ReassigningReferenceExample(String myObject)
    {
        this.myObject = myObject;
    }
    public static void main(String args[])
    {
        ReassigningReferenceExample testObject1 = new
        ReassigningReferenceExample("testObject1");
        ReassigningReferenceExample testObject2 = new
        ReassigningReferenceExample("testObject2");
        // testObject1 now refers to testObject2
        testObject1 = testObject2;
        // Requesting garbage collection
        System.gc();
    }
    @Override
    protected void finalize() throws Throwable
    {
        // following line will confirm the garbage collected method name
        System.out.println("Garbage collection is successful for " + this.myObject);
    }
}
```

Since the reference ID of the first object, `testObject1`, is eventually being used to reference the second object, `testObject2`, the first object becomes unreachable and is suitable for garbage collection, as shown in the code above. The output of the code will be as follows.

Garbage collection is successful for testObject1

The example above shows the importance of using the right reference ID at the right time to ensure that objects or variables that are crucial for the successful execution of your code, application, or program are not uselessly lost.

15.6.3 Nullified Reference Variables

Another extremely effective method to make an object suitable for garbage collection is making all of the variables that reference to it NULL. When this is done, you will have a scenario similar to the one mentioned above, and the object will have no references to it, essentially making it useless or unreachable. As soon as the object becomes unreachable, it is suitable for garbage collection, and the garbage collector can be called to remove it from the heap.

The following is an example code that shows how nullifying the reference variables of an object can make it unreachable and eligible for garbage collection:

```

package java11.fundamentals.chapter15;
public class NullifiedReferenceVariablesExample {
    private String myObject;
    public NullifiedReferenceVariablesExample(String myObject)
    {
        this.myObject = myObject;
    }
    public static void main(String args[])
    {
        NullifiedReferenceVariablesExample testObject1 = new
        NullifiedReferenceVariablesExample("testObject1");
        // Setting testObject1 to Null will qualify it for the garbage collection
        testObject1 = null;
        // Requesting garbage collection
        System.gc();
    }
    @Override
    protected void finalize() throws Throwable
    {
        // following line will confirm the garbage collected method name
        System.out.println("Garbage collection is successful for " + this.myObject);
    }
}

```

Since there is no longer any reference to `testObject1` and its reference variable was made `NULL`, `testObject1` is no longer reachable in the code and becomes suitable for garbage collection. When the garbage collector is called, it finds `testObject1` without any reference and removes it from the heap.

The output of the code above will be as follows.

Garbage collection is successful for testObject1

15.6.4 Anonymous Objects

Anonymous objects can be used in Java to call methods. However, what distinguishes anonymous objects from regular objects in Java is that anonymous objects do not have any reference IDs. As per the criteria, this makes anonymous objects the perfect candidates for garbage collection.

The following code is an example of a method being used on an anonymous object:

```

package java11.fundamentals.chapter15;
public class AnonymousObjectsExample {
    public static void main(String[] args) {

        System.out.println(new AnonymousObjectsExample().myMethod());

    }
    public String myMethod() {
        return "I love this book";
    }
}

```

The output of the code above will be as follows.

I love this book

Now that you understand how anonymous objects can be used to successfully call and run methods, here is an example of how garbage collectors can be used to remove anonymous objects from the heap.

```
package java11.fundamentals.chapter15;
public class AnonymousObjectsGarbageCollectionExample {
    String myObject;
    public AnonymousObjectsGarbageCollectionExample(String myObject)
    {
        this.myObject = myObject;
    }
    public static void main(String args[])
    {
        // Anonymous Object is being initialized without a reference id
        new AnonymousObjectsGarbageCollectionExample("testObject1");
        // Requesting garbage collector to remove the anonymous object
        System.gc();
    }
    @Override
    protected void finalize() throws Throwable
    {
        // following line will confirm the garbage collected method name
        System.out.println("Garbage collection is successful for " + this.myObject);
    }
}
```

Since there is no any reference to the anonymous object, the garbage collector will successfully remove it from the heap. The output of the code above will be as follows.

Garbage collection is successful for testObject1

15.7 | JEP 318 – Epsilon: A No-Op Garbage Collector



Interesting changes were proposed to the way garbage collection was approached in Java in a March 2018 update. For Java 11 version, the goal was to develop a garbage collection mechanism that took care of memory allocation but did not quite use a significant methodology for the reclamation of reusable memory from the heap. In this update, once the heap for Java was exhausted, the JVM would shut down. This garbage collector is called no-op garbage collector, which is also known as Epsilon.

Since the proposition for the update was based on the premise that the garbage collector would not reclaim memory from the heap, the no-op garbage collector would encourage the creation of ultra-performing applications that do not have any garbage or can do without a garbage collector for memory reclamation. The no-op garbage collector is intended to be simultaneously available with other garbage collectors and will not come into effect unless it is activated explicitly.

While this comes as an interesting update for the public, developers and researchers are particularly interested in the viability and practicality of the no-op garbage collector, considering how memory allocation and garbage collection work in Java. Moreover, since the no-op garbage collector will be simultaneously available with the other garbage collectors, users will be able to benefit from the no-op garbage collector or Epsilon collector as a control variable to gauge the performance of the remaining available garbage collectors.

To test the efficiency and effect of garbage collectors on the performance and speed of an application, variable configurations of garbage collectors can also be used simultaneously with the no-op garbage collector with the same workload. By doing so, garbage collector developers will not only be able to see how the performance of applications differs based on the configuration of the garbage collector in a controlled environment, but they will also be able to understand how garbage collectors work in a more isolated manner.

The Epsilon garbage collector or no-op garbage collector can prove to be highly beneficial for a limited number of applications and libraries that do not produce any garbage. Since the presence of a garbage collector is essentially not necessary, removing the overhead of the garbage collector can improve the efficiency of the application or library that is being used. However, to create a library that supports the Epsilon garbage collector, a number of factors including the library's memory management aspect without the use of a garbage collector must be taken into consideration. Since implementation of Epsilon garbage collector will essentially leave the application with no mechanism for reclamation of memory, it will have to be ensured that garbage is either non-existent, or minimal to the extent that the memory of the application does not run out.

When talking about the Epsilon garbage collector, it is important to consider the risks and benefits of implementing a no-op garbage collector and weighing them against the problems that will manifest in order to achieve a state of no or minimal garbage in any application or program. While there are a considerable number of difficulties in reaching a no garbage state considering how memory management works in Java, a few aspects of garbage collection need to be discussed to get an idea of how achieving such a state may be possible.

There are two major mechanisms used by JVM for memory management in Java. While most memory management operations are done through the heap, the stack is equally important in order to create an application that is equal parts memory-efficient and functional. The presence and use of both the heap and stack is the primary reason why there are two different types of errors when it comes to memory management – OutOfMemoryError and StackOverflowError.

The stack is only visible and used by threads that are running at a certain point in time, that too during the execution of its particular method. When the execution of the thread that is using the stack is complete, it leaves the stack and memory is automatically freed without the use of a garbage collector. It is the memory on the heap that needs to be checked by a garbage collector to see whether or not it can be cleaned to add value to the application or program that is being used. However, it is important to note that all 8 primitive data types go directly on the stack, which makes it possible for the application or program to run efficiently without the use of a garbage collector. If the Epsilon garbage collector is to be implemented, it is suggested that primitive data types be used for the majority of purposes to ensure that there is no any additional strain or reason for the presence of a conventional garbage collector.

Contrary to popular belief, objects can also be created without the use of a garbage collector. This means that fully-functional applications and programs can still be created with the Epsilon update with just a little additional effort.



Can we avoid OutOfMemoryError?

Summary

With this discussion about the Epsilon update available, we close the topic of garbage collection and how it works in the Java language. It is assumed that the codes, examples, and scenarios provided have helped you understand the ins and outs of garbage collection in the Java language and will give you an idea of the different ways in which garbage collection can be approached. In this chapter, we have learned the following concepts:

1. How garbage collector works.
2. What is the use of garbage collection?
3. Importance of memory management.
4. How to make objects eligible for garbage collection.
5. What are the latest updates in garbage collection?

In the Chapter 16, we will learn about String and I/O operations. We will also explore Java's file management capabilities and tools available to read, write, and manipulate file content.

Multiple-Choice Questions

1. At the time of object destruction, _____ method is utilized to execute some action.

(a) delete()	(b) finalize() (c) main() (d) None of the above
--------------	---

2. _____ requires the highest memory.
- Class
 - JVM
 - Stack
 - Heap
3. Where does the new object memory get allotted?
- JVM
 - Young Space
 - Old Space
 - Young or Old Space, depending on space availability
4. _____ is a garbage collection algorithm which has two phases operation.
- Space management tool
 - Sweep model
 - Cleanup model
 - Mark and sweep model
5. Which of the following is not a Java Profiler?
- Jconsole
 - JVM
 - Jprofiler
 - Eclipse Profiler

Review Questions

- How does garbage collection work in Java?
- How do we make objects eligible for garbage collection?
- How can you instruct the garbage collector to initiate the garbage collection process?
- What is the inner working of the garbage collector?
- Why is memory management important?

Exercises

- Create a diagram that shows how objects become eligible for garbage collection.
- Write a program that initializes a lot of objects in a loop and observe how much time it takes to crash the program.
- Write a program that can generate stack overflow error. Document the findings.

Project Idea

Create a voting system program that can collect the entire list of the voters from all around the country and allow them to vote. This program must validate the identity of the users.

Calculate the number of votes. Make sure you use good garbage collection practices so the program will not crash due to memory management issue.

Recommended Readings

- Benjamin J. Evans, James Gough, and Chris Newland .2018. *Optimizing Java: Practical Techniques for Improving JVM Application Performance*. O'Reilly Media: Massachusetts
- Erik Ostermueller. 2017. *Troubleshooting Java Performance: Detecting Anti-Patterns with Open Source Tools*. Apress Media: New York
- Oracle Technetwork: Java Garbage Collection Basics – <https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

Strings, I/O Operations, and File Management

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- All about strings.
- Various string operations such as concatenation and split.
- StringBuffer and StringBuilder.
- I/O operations.
- InputStream and OutputStream.
- File management.
- Access files and manipulation of files.

16.1 | Introduction

The String class of Java programming language is one that interests many due to its unique characteristics. Unlike the strings of other programming languages such as C++ or C, which are essentially only arrays of chars, strings in Java comprise immutable sequences of Unicode characters that make them unique in many ways.

To make the creation and manipulation of strings easier than it is in most programming languages, the String class in Java offers several different features. Owing to these features, there are a few different techniques that can be used to create and modify strings according to one's requirements or preferences and to ensure that they add value to the application, program, or block of code where they are being used. The creation of strings, for example, can be done using either a string literal, or by using a constructor and calling it for the creation of a String instance. Both techniques will be explained in detail in subsequent sections in this chapter.

16.2 | Role of Strings in Java

Before we discuss the different reasons and ways in which Strings can be used in Java, it is important examine a few facts that every programmer of the Java language should know about strings. We already know that Strings in Java are unlike strings in other languages such as C and C++. They are not simply arrays of chars, and there is a lot that makes the Strings in Java unique.

One main feature that makes Strings in Java important is concatenation. With just a simple “+” sign, you can easily concatenate the contents of two or more strings – something which is not possible with other objects in the Java language, such as Circle or Point. Additionally, since the contents of Strings in Java is as immutable as mentioned above, modifications cannot be made to the contents of any string that is being used in any application, program, or block of code. This means that functions like `toLowerCase()` or `toUpperCase()` will create an entirely new string instead of modifying or changing the contents of the current string, and this new string will then be returned.

The use of null characters for the termination of strings in other languages such as C or C++ is another way in which Strings of Java are different. Strings in Java are objects that are backed by character arrays. If you wish to view the contents of the String in Java in the form of the character array that represents it, you can use the `toCharArray()` method of the String class.

Comparison of Strings in Java is also far easier than it is in other programming languages. Instead of following a lengthy process for the comparison of strings, all you need to do is use the `equals()` method for the comparison of two strings that are being used in any program, application, or block of code. This is possible because the String class of Java overrides the `equals()` method, making comparison of strings extremely easy, convenient, and hassle-free.

Similarly, searching for substrings within a string of Java is also far easier than in other languages. With the help of regular expressions and simple methods such as `indexOf()` and `lastIndexOf()`, parts of strings can be searched for and values

returned if a match is found. Strings can also be trimmed and split into multiple other strings using regular expressions and used separately for a variety of purposes in any application, program, or block of code in Java.

Now that you are aware of the variety of benefits and features that the String class and strings in Java language offer, we will discuss the String class and how strings work in Java programming.

You have learned about how memory is managed by garbage collection in Java language in Chapter 15. Here is a quick recap. There are two major entities that are used for memory management in Java – the heap and the stack. The stack is used for the execution of operations and processes as they are called in the block of code, program, or application. On the other hand, the heap has more to do with storage of contents that are required for the effective running and execution of the code, program, or application.

But what does that have to do with strings? The answer is not exactly simple or straightforward by any means. Strings and the String class of Java programming language are given special treatment, and any string literals that are used in the programming language are assigned a special storage space in the heap memory known as *string constant pool*. Whenever string objects are created using string literals in Java, these objects are stored in the string constant pool. On the other hand, when string objects are created using the new keyword, they are treated just like other objects and are sent to the heap for storage purposes. The following is an example of how string objects are created using string literals:

```
package java11.fundamentals.chapter16;
public class LiteralExample {
    public static void main(String args[]) {
        String message = "Hello World! I love Java";
        System.out.println(message);
    }
}
```

The String Object of the string message shown below with contents “Hello World! I love Java” has been created with the help of a string literal. It will go in the string constant pool instead of being sent to the heap memory like other string objects.

Hello World! I love Java

Similarly, String Objects can also be created using the new keyword as follows:

```
package java11.fundamentals.chapter16;
public class KeywordExample {

    public static void main(String args[]) {
        char[] javaArray = { 'I', ' ', 'L', 'O', 'V', 'E', ' ', 'J', 'A', 'V', 'A' };
        String javaString = new String(javaArray);
        System.out.println(javaString);
    }
}
```

The example above shows how a string object can be created using the new keyword. See the output below.

I LOVE JAVA

As mentioned earlier, string objects that are created using this method are treated like normal objects and are sent to the heap, where they are stored along with other objects and variables that are important for the execution of codes or programs.

What most people do not know about the string constant pool is that pool space is allocated to objects depending on the content of the string object in question. This means that when objects are sent to the string constant pool, they are checked to ensure that there are not any two objects that have the same content.

Whenever a new object needs to be created using string literal, the Java virtual machine (JVM) goes through the content of the object that the user wants to create, and then double-checks the content of the objects that are already available in the pool. If an object with content that is the same as the one that needs to be created already exists in the pool, the reference of this object is returned and the new object is not created. The new object will only be created if the content in it is unique and distinct.

However, this is not the case when the new keyword is used for the creation of a new string. If you attempt to create a new string using the new keyword, it will be created whether or not it contains the same contents as an existing string. This shows that two string objects present in the heap memory can have the same contents, but that is not the case with string objects present inside the string constant pool. This can also be proven using the `==` operator, which only returns as true if the physical address of both objects being compared is the same.

```
package java11.fundamentals.chapter16;
public class StringObjectComparison {
    public static void main(String[] args) {
        // The following strings are being created using literals
        String literal1 = "xyz";
        String literal2 = "xyz";
        System.out.print("Comparison using == operator for literals : ");
        System.out.println(literal1 == literal2); // The output of this line of code will
be true
        // The following two strings are being created using the new operator
        String keyword1 = new String("abc");
        String keyword2 = new String("abc");
        System.out.print("Comparison using == operator for objects : ");
        System.out.println(keyword1 == keyword2); // The output of this line of code will
be false
    }
}
```

The above program creates the following output.

```
Comparison using == operator for literals : true
Comparison using == operator for objects : false
```

Since the two string objects created using string literals having the same contents will also have the same physical address, the output of the first comparison returns as “true”. On the other hand, even though the contents of the string objects created using the new keyword are also the same, this second comparison will return as “false” because each of these string objects will have different physical addresses in the heap. This further proves that two objects cannot have the same contents if they need to be stored in the string constant pool, whereas it is completely normal for them to coexist in the heap memory. See Figure 16.1 to understand how String objects are located on Java heap and how they are compared to each other.

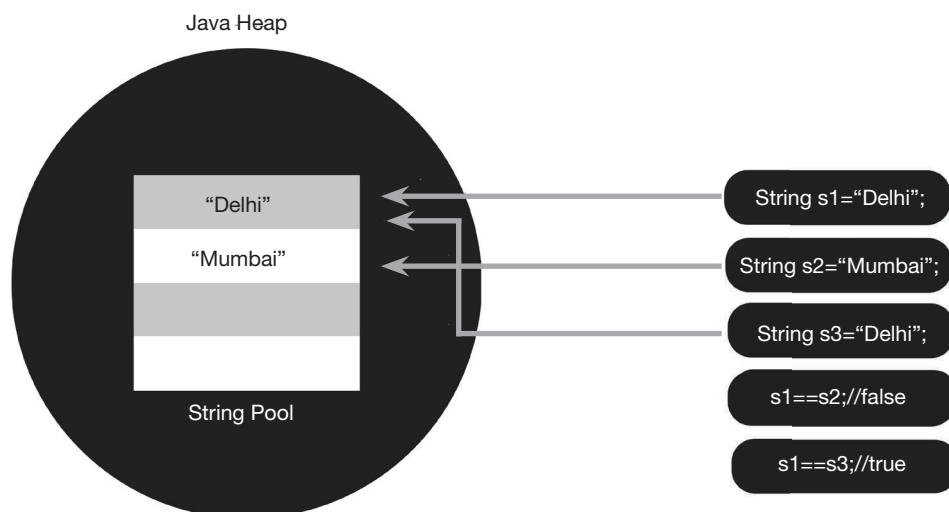


Figure 16.1 String Pool on Java heap and `==` Comparison.

Understanding the difference between the strings that go on the heap and those that go on the stack is imperative in order to understand which objects can potentially be garbage collected more easily, once they are no longer in use. There are essentially four major types of references that exist in Java:

1. Strong references.
2. Weak references.
3. Soft references
4. Phantom references.

For the sake of explanation, consider the following example:

```
Greeting hello = new Greeting();
```

In the line of code above, “hello” acts as a strong reference to the `Greeting()` object.



Can the garbage collector collect a String object?

In case an object does not have a strong reference (and only has a soft reference), there is a high possibility of the memory of said object being recollected in case the garbage collector needs additional memory for operations. On the other hand, if an object only has a weak reference assigned to it, the garbage collector will reclaim the memory of this object in the next cleaning phase, regardless of whether or not enough memory already exists.

If an object does not have a strong, weak, or soft reference, then the reference that it has is often called *phantom reference*. Unable to be accessed directly, these references are not known to many programmers or developers of Java, which makes them particularly interesting. Another important fact is that whenever the `get()` method is used on phantom references, they always return as null.

The most popular and powerful types of references – strong references – are used extremely common for programs, blocks of code, and applications that are developed using Java. Objects can be easily created in Java language and assigned references. It is important to note that whenever an object has a strong reference, it can never be garbage collected.

Since strings are given special treatment in Java, the same holds true when it comes to garbage collection of string objects that are used in blocks of code, programs, or applications created using the language. As you already know, every time a thread is created and started, it has its own stack memory. Here, it is important to note that even if an object is present in the heap, but is no longer being referenced by the stack, it becomes eligible for garbage collection. Even if an object in the heap has strong references to other objects present within the heap, they become eligible for garbage collection and will eventually be removed or deleted if they do not have a reference from the stack.

Here are a few facts regarding the garbage collection process and how it really works:

1. Garbage collection is an automatic process in Java. What this means is that starting the process is at the discretion of the JVM.
2. Garbage collection is actually an expensive process. This is because the running of the garbage collector essentially puts all the other threads of the application, program, or block of code on hold, until the garbage collection process is completed.

More complex than simply calling a method and freeing up memory, the garbage collection process essentially entails the use of the mark-and-sweep mechanism that helps JVM decide which objects need to be kept alive for the program, block of code, or application to run and be executed effectively. This helps us understand that even though garbage collection essentially works automatically in Java, certain objects and processes can still be left alive to ensure that the quality, efficiency, and/or performance of the application, program, or block of code in use is not being compromised in any way.



What are some of the most common string operations in the Java programming language?

16.3 | Types of String Operations

There is a wide variety of operations that can be applied to strings when used in Java language. As mentioned earlier, strings in Java can be concatenated and split, and they can also be formatted. Since these operations are extremely important and commonly used for the creation of codes, we will explain them in detail along with examples of codes to help you understand how exactly each operation can be used in your own applications or programs.



16.3.1 Concatenation

Concatenation is one of the most commonly used operations when it comes to strings in Java language. Concatenation in Java is the name of the process that is used to combine the contents of two or more strings to create a new string. There are two major methods that can be used to concatenate two strings in Java:

1. The first method involves the use of the “+” operator.
2. The second method involves the use of the concat() method of the String class.

16.3.1.1 Concatenation Using the Addition “+” Operator

Concatenation using the addition “+” operator is the most commonly used technique to add the contents of two strings in Java. It is important to note that whenever you want to add two or more string literals together, they should be within double quotes.

For instance, if you want to combine the strings “Hello” and “people of the world”, you should write the line of code as follows:

```
"Hello" + " people of the world!" //result: Hello people of the world!
```

Here, it is also important to note that for your sentence to read properly, you should ensure that you properly add spaces in the double quotes.

You can also use the concatenation option for printing outputs in Java. The code to add the contents of the same two strings as mentioned above and print the output is as follows:

```
System.out.println("Hello" + " people of the world!"); //the output Hello people of the world! will be printed on the screen.
```

1. **Combination of strings on more than one line:** When it comes to string literals, Java does not accommodate the contents of a string to span multiple lines. This is another area where the concatenation option can come in handy.

With the help of the concatenation operation using the “+” operator, you can create a string literal that spans multiple lines, shown as follows:

```
String PopularQuote = "An eye for an eye" +
" will make the whole world blind."
```

2. **Concatenating variable objects:** While the “+” sign is often used as an arithmetic operator, the rules change considerably in case one of the operands with the “+” sign is a string. In such a case, the other operand is also converted into string form to ensure that it makes sense when concatenated with the operand of String type. Let us take a look at an example:

```
float weight = 50.0;
System.out.println("My weight is " + weight);
```

The output for the line of code written above would be:

```
My weight is 50.0
```

In the example above, weight is a float variable, so the “+” operator will first convert the operand to String type and then concatenate the two strings, as it normally would. Even though it is not visible to the end user or programmer who wrote the block of code, the conversion from float type to String type is done by calling the `toString()` method. This shows that concatenation operation does a lot more than just combine the contents of two strings, and that background operations will also be done if needed.

QUICK CHALLENGE

Consider two strings s1 and s2. Assign values to each like s1=“ABC” and s2=“XYZ”. Now try using “*” operator on those two and note down the result.

16.3.1.2 Concatenation Using the concat() Method

The second technique that can be used for concatenation of two strings in Java is by using the `concat()` method. Whenever the `concat()` method of the `String` class is used, the method is applied to the first string that needs to be added to form the result, and the second string that needs to be concatenated is taken as a parameter. Let us see the following example:

```
public String concat (String myStr)
```

The line of code above shows how `concat()` method takes the second string as a parameter to add, or concatenate, it with the first string.

The following is another example of how an entire block of code is written to concatenate two strings using the `concat()` method of the `String` class:

```
package java11.fundamentals.chapter16;
public class StringConcatExample {
    public static void main(String args[]) {
        String myStr = "My Favourite Programming Language";
        myStr = myStr.concat(" is Java");
        System.out.println(myStr);
    }
}
```

The output of the block of code above is shown below.

My Favourite Programming Language is Java

This shows that the `concat()` method works in a way that is different from the “+” operator. There are quite a number of other differences between these two techniques.

1. While the “+” operator can be used to concatenate objects of variable types, the `concat()` method can only be used to combine objects of the `String` type. What this means is that the `concat()` method of the `String` class will only work effectively in case it is called on a variable of `String` type and has a parameter of the `String` type that needs to be concatenated.

This makes the `concat()` method a lot more limited than the “+” operator. The latter is a lot more convenient and hassle-free since it can convert variables of a number of data types into `String` type efficiently and effectively, allowing this operator to offer a wider range of benefits and usages than the `concat()` method.

2. The second major difference between these two methods is that an exception is thrown by the `concat()` method if the object that is entered as a parameter has a null reference. This means that the `concat()` method of the `String` class throws a `NullPointerException` whenever a parameter has a null reference. On the other hand, the “+” operator treats the second operand as a null string and still concatenates it with the first string or operand.
3. Unlike the “+” operator that can be used to concatenate multiple strings, the `concat()` method of the `String` class can be used for the concatenation of only two strings at a time.

Due to the reasons mentioned above, it goes without saying that the “+” operator is used more commonly for concatenation of strings in Java than the `concat()` method. However, since there are significant differences in the working of both of these techniques, the performance and efficiency of applications will also differ depending on the technique that is being used for concatenation.

16.3.2 Splitting Strings

Another extremely common operation that is performed on strings in Java is splitting. There are several different reasons why you might need to split a string into two or more parts, which is why it is important for you to understand the working behind the `split()` method of the `String` class.

The `split()` method of the `String` class in Java splits or divides the input string into multiple parts based on the regular expression that is entered as a parameter. The result of this operation is an array of strings that are divided according to the regular expression that was input as the parameter. In one variant of the `split()` method, you even have the option of entering your desired limit for threshold of the result that will be the output.

For the first variant of the `split()` method, the only thing that is required is the string that needs to be split. All other operations and workings will be done by the Java programming language itself.

The following is an example of how the `split()` method works without a limit:

```
package java11.fundamentals.chapter16;
public class StringSplitExample {
    public static void main(String args[]) {
        String myStr = "My Favourite Programming Language : Java";
        String[] arrOfStr = myStr.split(":");
        for (String piece : arrOfStr) {
            System.out.println(piece);
        }
    }
}
```

The result of the block of code given above as an example is shown below.

My Favourite Programming Language
Java

This shows that it is not necessary for one to enter any limits when using the `split()` method in the Java. In the block of code mentioned above, the String `myStr` was used as an example, and the string was split by the ":" as mentioned in the following line:

```
String[] arrOfStr = myStr.split(":");
```

When printed, the block of code gave us the desired result.

As mentioned earlier, the other variant of the `split()` method of the `String` class in Java requires the user to add a limit to the result. This limit is entered as an integer parameter. It is important to note that there are three types of values that the limit integer can take:

- 1.** Positive.
- 2.** Negative.
- 3.** Zero.

Since the value that you select for the limit has an effect on the result that you will get after the string is split, it is important to understand how the value of the result can potentially change according to the type of value that you select for the limit.

- 1. Positive:** In case the user selects a positive limit for the result, the pattern will be repeated a maximum of `limit - 1` times. Additionally, the length of the final array will not be more than the size of the string itself, and the last entry of the final array will contain the remaining part of the string after the pattern was last matched.
- 2. Negative:** In case the value for the limit entered by the user is negative, the pattern will be repeated as many times as possible. When the value for the limit is set as a negative integer, there will also be no limits on the size of the final resulting array.
- 3. Zero:** When the value of the limit variable is set as zero, the pattern will be repeated as many times as possible. Again, the final resultant array can be of any size. It is, however, important to note that empty strings will be discarded from the final result.

**QUICK
CHALLENGE**

Write an algorithm which can split any string without using the `split()` method. Print timestamp before and after the code to verify which method is faster.

The following is an example to help you understand how the limit works for the `split()` method:

```
package java11.fundamentals.chapter16;
public class StringSplitWithLimitExample {
    public static void main(String args[]) {
        String myStr = "I@love@java";
        String[] arrOfStr = myStr.split("@", 2);
        for (String piece : arrOfStr) {
            System.out.println(piece);
        }
    }
}
```

The output of the block of code written above will be as follows.



I
love@java

Since the limit was set as 2 and the string was to split at “@”, there are only two substrings in the result, which is seen in the output.

Similarly, there can also be multiple different characters that can be entered as the parameter at which the string will be split. Each of these characters will have to be explicitly mentioned when they are entered as parameters. The following is an example of how this works:

```
package java11.fundamentals.chapter16;
public class StringSplitOnMultipleCharactersExample {
    public static void main(String args[]) {
        String myStr = "My, Favourite @Programming?Language.Java";
        String[] arrOfStr = myStr.split("[, ?.@]+");
        for (String piece : arrOfStr) {
            System.out.println(piece);
        }
    }
}
```

The result of the block of code written above will be as follows.



My
Favourite
@Programming?
Language
Java

Since no limits were set for the maximum number of results, the program showed 5 different results, each of which were separated by one of the characters mentioned in the regular expression.

All of the examples mentioned above show that by playing around with your regular expressions and the limits that you set for the number of results that you need, you can modify the types of results that you will get. Moreover, you can also decide the number of substrings that your input string will be divided into.

As mentioned at the beginning of the chapter, strings in Java language are immutable, which means that their contents cannot be changed or modified. This is the reason why new strings have to be created whenever an operation needs to be performed on any string. Fortunately for programmers and application developers of Java language, mutable strings have also been accommodated. Different options are now available for the manipulation of strings without burdening the machine too much, or producing excessive amounts of garbage.

Thanks to the `StringBuilder` and `StringBuffer` classes, manipulation of string objects in Java is far easier than it would be if only the `String` class existed. The `StringBuffer` and `StringBuilder` classes allow strings to easily be manipulated without the need for additional strings to be created. This way, you not only save on garbage that can affect the efficiency of your program, but the performance of your application, program, or block of code also gets much better than it was.

Since both `StringBuilder` and `StringBuffer` are mutable objects in Java, they offer multiple different manipulation options for the strings that are created. Some of the methods offered by these classes include the `insert()`, `delete()`, and `append()` methods that are commonly used for the manipulation of strings. While both the `StringBuffer` and `StringBuilder` classes are essentially used for the manipulation of strings that are created in Java, there are certain significant differences between the two. These will be discussed in detail in following section.

16.4 | **StringBuilder and StringBuffer Explained**



The primary reason why both `StringBuilder` and `StringBuffer` are used is for the manipulation of strings in Java language, which makes both of these classes mutable. These are unlike the more popular `String` class, which is used for the creation of strings in Java language. However, this is perhaps the only thing that both the `StringBuilder` and `StringBuffer` classes have in common.

Unlike the `StringBuilder` class, the `StringBuffer` class is thread safe. This means that this class accommodates and modifies data structures only in a way in which they are guaranteed to be executed safely by multiple threads simultaneously. Since the Java language supports the idea of concurrency, this feature of the `StringBuffer` class is of particular interest to developers and programmers who try to incorporate concurrency in their programs and applications. The `StringBuffer` class also contains the `insert()` and `append()` methods, which are popular among programmers who want to manipulate strings in multi-thread environments. Since a wide variety of string operations in Java occur in single-thread environments, the `StringBuilder` class was created without the thread safety option.

The fact that the `StringBuilder` class does not support concurrency or work in multi-threading environments is also the major reason why the `StringBuilder` class is considerably faster than the `StringBuffer` class. Also, the “+” operator that is used for concatenation of strings in Java also uses either the `StringBuilder` class or the `StringBuffer` class internally to perform efficient addition or combination of two or more strings in any application, program, or block of code. Moreover, since `StringBuilder` was introduced in a later version of Java, most of the problems and shortcomings of the `StringBuffer` class have been overcome in the `StringBuilder` class.

QUICK CHALLENGE

Give a scenario where you would consider `StringBuilder` over `StringBuffer`.

Additionally, since the `StringBuilder` class does not support the idea of synchronization, its performance, efficiency, and speed are considerably different compared to the `StringBuffer` class. Synchronization does not only affect processing power negatively, but also accounts for additional overhead that is completely useless.

To validate the claims made in this section that compares the `StringBuffer` and `StringBuilder` classes, let us see the example of a program that repeatedly performs the `insert()` and `append()` methods on objects of both of these classes. With the help of this program and different test values, we will show how the performance of both of the classes differs and to what extent.

```

package java11.fundamentals.chapter16;
import java.util.GregorianCalendar;
public class StringBufferVsStringBuilderExample {
    public static void main(String[] args) {
        System.gc();
        StringBuffer myStrBuff = new StringBuffer();
        StringBuilder myStrBuild = new StringBuilder();
        runStringBuilder(myStrBuild);
        // Request Garbage Collection to clear the memory
        System.gc();
        runStringBuffer(myStrBuff);
    }

    private static void runStringBuilder(StringBuilder myStr) {
        long begin = new GregorianCalendar().getTimeInMillis();
        long initiateMemory = Runtime.getRuntime().freeMemory();
        for (int j = 0; j < 50000; j++) {
            myStr.append(": " + j);
            myStr.insert(j, "Hello");
        }
        long finish = new GregorianCalendar().getTimeInMillis();
        long stopMemory = Runtime.getRuntime().freeMemory();
        System.out.println("Time Taken for String Builder Append Insert:" + (finish - begin));
        System.out.println("Memory used String Builder Append Insert:" + (initiateMemory - stopMemory));
    }

    private static void runStringBuffer(StringBuffer myStr) {
        long begin = new GregorianCalendar().getTimeInMillis();
        long initiateMemory = Runtime.getRuntime().freeMemory();
        for (int j = 0; j < 50000; j++) {
            myStr.append(": " + j);
            myStr.insert(j, "Hello");
        }
        long finish = new GregorianCalendar().getTimeInMillis();
        long stopMemory = Runtime.getRuntime().freeMemory();
        System.out.println("Time Taken for String Buffer Append Insert:" + (finish - begin));
        System.out.println("Memory used String Buffer Append Insert:" + (initiateMemory - stopMemory));
    }
}

```

The above program produces the following result.

```

Time Taken for String Builder Append Insert:333
Memory used String Builder Append Insert:1603872
Time Taken for String Buffer Append Insert:318
Memory used String Buffer Append Insert:1175040

```

As mentioned above, the value of all variables was changed multiple times, and the program was repeated with each modification to check how the results varied. The value of j was changed from 1000 to 50,000, and the program was repeated multiple times for both `StringBuilder` and `StringBuffer`. However, there was not too great a difference in the performance of both these classes.

Since the `insert()` method requires a lot of memory and produces plenty of garbage, the same test can also be conducted on both the `StringBuilder` and `StringBuffer` classes without this method to get a better understanding of how the efficiency and performance of both these classes differ.

Additionally, since larger numbers of repetitions will give users and readers a better understanding of how much the results differ. The following is another code which is repeated 50,000,000 times to put things into perspective.

```
package java11.fundamentals.chapter16;
import java.util.GregorianCalendar;
public class StringBufferVsStringBuilderWithoutInsertExample {
    public static void main(String[] args) {
        System.gc();

        StringBuffer myStrBuff = new StringBuffer();
        StringBuilder myStrBuild = new StringBuilder();

        runStringBuilder(myStrBuild);

        // Request Garbage Collection to clear the memory
        System.gc();
        runStringBuffer(myStrBuff);
    }

    private static void runStringBuilder(StringBuilder myStr) {
        long begin = new GregorianCalendar().getTimeInMillis();
        long initiateMemory = Runtime.getRuntime().freeMemory();
        for (int j = 0; j < 50000000; j++) {
            myStr.append(": " + j);
        }
        long finish = new GregorianCalendar().getTimeInMillis();
        long stopMemory = Runtime.getRuntime().freeMemory();
        System.out.println("Time Taken for String Builder Append:" + (finish - begin));
        System.out.println("Memory used String Builder Append:" + (initiateMemory -
stopMemory));
    }

    private static void runStringBuffer(StringBuffer myStr) {
        long begin = new GregorianCalendar().getTimeInMillis();
        long initiateMemory = Runtime.getRuntime().freeMemory();
        for (int j = 0; j < 50000000; j++) {
            myStr.append(": " + j);
        }
        long finish = new GregorianCalendar().getTimeInMillis();
        long stopMemory = Runtime.getRuntime().freeMemory();
        System.out.println("Time Taken for String Buffer Append:" + (finish - begin));
        System.out.println("Memory used String Buffer Append:" + (initiateMemory -
stopMemory));
    }
}
```

The above program produces the following result.

Time Taken for String Builder Append:2200
Memory used String Builder Append:-533019072
Time Taken for String Buffer Append:2401
Memory used String Buffer Append:-14155776

When the test was repeated without the `insert()` method as mentioned earlier, a considerable difference was seen in the amount of time that was taken for the execution of the program to be completed by both the `StringBuilder` and `StringBuffer` classes. With the help of this updated example, it is evident that the `StringBuilder` class can perform better than the `StringBuffer` class even when multi-threading is not used.

Now that you know how string operations work in Java and how the `String`, `StringBuilder`, and `StringBuffer` classes differ from each other, we look into other significant areas of the Java programming language including file operations and I/O operations.

In the sections that follow, we will talk extensively about Java File Class, and how methods and operations can be performed for optimal results and maximum efficiency.

16.5 | Java I/O



Ever wondered why you have to write “`import java.io`” at the beginning of most, if not all, of your Java codes? We will answer this question by first understanding Java I/O. As the name suggests, Java I/O is what is used at the back end to process all input and output operations to make your programs work seamlessly and efficiently.

All this is done with the help of streams! Streams are used in Java language to not only expedite the input and output process, but to also make all operations and processing seamless in order to provide optimal results. Additionally, the `java.io` package comprises multiple classes, each of which can prove to be beneficial for a number of different reasons, helping you complete all operations efficiently and effectively. The console for Java comes with three built-in byte streams to make processing easier. These streams are:

1. **System.out:** This is the standard output stream that is used for operations in Java.
2. **System.in:** This is the standard input stream that is used for operations in Java.
3. **System.err:** This is the standard error stream that is used for input output operations in Java.

The following are a few sample codes to show you how each of these streams work to make the input and output process in Java more efficient:

```
package java11.fundamentals.chapter16;
import java.util.Scanner;
public class InputOutputProcessExample {
    public static void main(String args[]) {
        // Following code will create scannerObj object of Scanner class
        Scanner scannerObj = new Scanner(System.in);
        System.out.println("Enter the name of the student");
        // Below line of code ensures that data will be input as string by default
        String studentNAME = scannerObj.next();
        System.out.println("Enter the roll number of the student");
        // Below line of code ensures that data will be input as int by default
        int studentRollNumber = scannerObj.nextInt();
        System.out.println("Enter the marks that the student obtained");
        // Below line of code ensures that data will be input as float by default
        float studentMarks = scannerObj.nextFloat();
        System.out.println("-----Student Report Card-----");
        System.out.println("Student Name:" + studentNAME);
        System.out.println("Student Roll No.:" + studentRollNumber);
        System.out.println("Student Marks:" + studentMarks);
        // Following code is needed to avoid resource leak
        scannerObj.close();
    }
}
```

The above program produces the following result.

```

Enter the name of the student
Mayur
Enter the roll number of the student
17
Enter the marks that the student obtained
100
-----Student Report Card-----
Student Name:Mayur
Student Roll No.:17
Student Marks:100.0

```

After each line of code where an input is requested from the user, the user will be allowed to enter a value according to the data type that is set as default.

To help put things in perspective, here is a line of code that explains how the System.err stream can be used:

```
System.err.println("This is an error message");
```

Now that you understand how the System.in, System.out, and System.err streams can be used, we will discuss the OutputStream and InputStream classes, their most popular methods, and how they differ. See Figure 16.2 to understand the role of InputStream and OutputStream in a Java application.

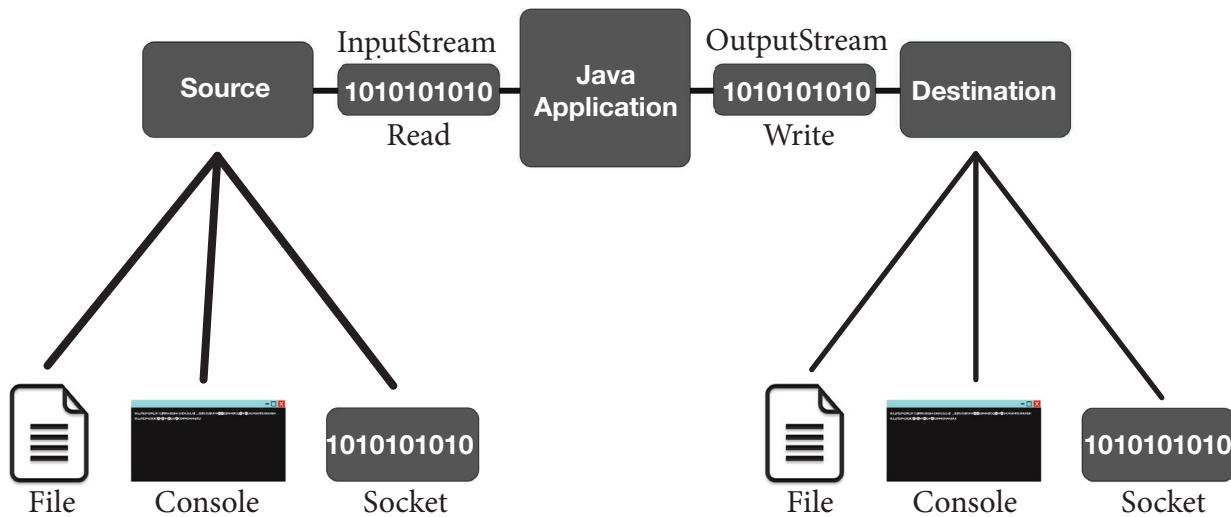


Figure 16.2 Role of InputStream and OutputStream.

16.5.1 InputStream

One of the major classes of the java.io package is the InputStream class. It provides users with a mechanism to help data be input into Java programs and be read without any problems. An abstract super class by definition, the InputStream class provides programmers and developers with the right tools to not only read bytes of data in singular and array form, but to do so selectively with data streams, mark locations within data streams, determine the number of bytes that are available to be read, and reset the current position within a stream of data.

Here, it is important to note that input streams in Java are opened as soon as they are created. The benefit with this feature is that you do not need to explicitly call the input stream whenever it is needed, and the console will take care of that bit automatically. When the input stream is no longer needed and all information that was requested from the user has already been entered, the `close()` method can be used to close the stream explicitly, or wait for the garbage collection process to be completed, which will automatically close and remove the input stream once it is no longer in use and no longer being referenced.

16.5.2 OutputStream

This abstract super class of Java language is essentially used to deal with all outputs in an efficient and effective manner. In addition to providing users with the mechanism and tools to write bytes of data and arrays of bytes, the OutputStream class acts as an interface that is used by multiple other classes to make processing of data more convenient and hassle-free. Much like input streams, output streams can also be closed explicitly by using the `close()` method, or by garbage collection.

Now that you know how both classes differ from each other, we will discuss some of the most popular methods of each of these classes to help you get a better understanding of how they can be used.

16.5.3 Methods of OutputStream

There are several different methods of the OutputStream class, each of which serves a unique purpose and offers a unique set of benefits. We will discuss a few of these methods and explain how they differ from each other.

1. The public void `write(int)` method is the method of the OutputStream class that is used to write a single byte to the output stream that is currently in use. It should be noted that this method cannot be used for arrays of bytes.
2. The public void `write(int[])` method is used for writing an array of bytes to the output stream that is currently in use. The difference between the public void `write(int)` and public void `write(int[])` is also evident from the parameters that each of these methods allow. Since the second method has int[] or integer arrays as a parameter, it goes without saying that this method should be used to write an array of bytes to the output stream that is currently in use.
3. The third method of the OutputStream class is the `flush()` method. As the name suggests, this method flushes the output stream that is being used at any given point of time.
4. The last method, `close()`, is used to close the output stream that is currently being used, removing all references and making it eligible for garbage collection.

16.5.4 Methods of InputStream

The following methods are part of the InputStream class and can be used to input data to be used by the program or application.

1. The first method is the public abstract int `read()` method. It is used to read the next available byte of data in the input stream. Whenever the method reaches the end of a file or stream, it returns -1 as an indication that there is no longer anything that can be read.
2. The second method is the public int `available()` method. It shows the user an approximate number of bytes that can still be read from the current input stream.
3. The third method is the public void `close()` method of the InputStream class works in the same way as the method of the same name in the OutputStream class, and is used to close the current input stream and make it eligible for garbage collection.



What permissions do you need on a server to use the File Management functionality?

16.6 | File Management in Java

File handling or file management is another area of Java that tends to interest a number of people, particularly due to the variety of operations that can be performed on files in the language. We will share some insights into file management in Java programming language.

The FileReader and FileWriter classes are of immense importance when it comes to file management in Java language. Inheriting the OutputStream class, the FileWriter class is used for the creation of files by writing characters. While certain assumptions are made by the constructors of this class, programmers and developers of Java are free to specify values by themselves by creating their own constructors. Some of the constructors of the FileWriter class are as follows:

1. **`FileWriter(File, File)`:** As the name suggests, this constructor creates a FileWriter object when a File object is input as a parameter.
2. **`FileWriter(String filename)`:** Since the parameter for this constructor is a String, the constructor will create a FileWriter object whenever a file name is input as a parameter.



Here are some of the methods that are available through the `FileWriter` class:

1. `public void write (int c)`: This method is used to write a single character into the stream that is being created.
2. `public void write(char[] stir)`: The character array that needs to be inserted to the output stream will be input into this method as a parameter.

Inheriting the `InputStreamReader` class, the `FileReader` class is used to read data from any file – one character at a time. It is important to note that the `FileReader` class can only be used to read data in the form of characters, while the `FileInputStream` class is used to read data in the form of raw bytes. Here are some of the constructors of the `FileReader` class:

1. `FileReader(File,File)`: As evident from the parameters of this constructor, a `FileReader` object is created when a `File` object is inserted as a parameter.
2. `FileReader(String filename)`: This constructor creates a `FileReader` object given that the name of the file that needs to be read from is input as a parameter.

Here are some of the methods that are available through the `FileReader` class:

1. `public int read()`: This method is used to read a single character from the stream that is available.
2. `public int read(char[] cbuff)`: This method is used to read characters into an array.



Can you read a file from a remote server?

Summary

With this, we have completed the chapter on Strings and how they work in Java language. We have also described in detail about input and output in Java programming language, and the methods and operations that are available to make processing more efficient and optimized.

In this chapter, we have learned the following concepts:

1. String operations and its usage.
2. Various string functions such as concatenation and split.
3. Differences between `StringBuffer` and `StringBuilder`.
4. Various I/O operations.
5. File management with file reader and file writer.

In Chapter 17, we will learn about data structures and its types, such as primitive and non-primitive data structures, and how to use them in a program.

Multiple-Choice Questions

1. Which of the following sentences is false about `String` in Java?
 - (a) We can extend `String` class like `StringBuffer` does it.
 - (b) `String` class is defined in `java.util` package.
 - (c) `String` is immutable in Java.
 - (d) `String` is thread-safe in Java.
2. Which of the following methods of `String` class can be utilized for testing strings for equality?
 - (a) `isequal()`
 - (b) `isequals()`
 - (c) `equal()`
 - (d) `equals()`
3. _____ class is used for reading characters in a file.
 - (a) `FileWriter`
 - (b) `FileReader`
 - (c) `FileInputStream`
 - (d) `InputStreamReader`
4. _____ method is used for reading characters in a file.
 - (a) `read()`
 - (b) `scan()`
 - (c) `get()`
 - (d) `readFileInput()`

5. _____ method is used for writing into a file.
- (a) putfile()
 (b) write()
 (c) writefile()
 (d) put()

Review Questions

1. What are various operations you can perform on String?
2. Is String immutable?
3. How to read and write content into a file?
4. How do we use StringBuilder?
5. What are the advantages of using StringBuilder over StringBuffer?
6. How do we use InputStream? Give one example.
7. How do we use OutputStream? Give one example.

Exercises

1. Write a program to test the difference between StringBuffer and StringBuilder.
2. Write a program to test all the string manipulations using various functions.
3. Create a program which writes content into a file and read from the file.

Project Idea

Create a job application portal that can accept an applicant's CV and process its data. The program should be able to

read the CV file and collect the applicant's name, address, education, employment details and show it on a page.

Recommended Readings

1. Oracle Tutorials: Regular Expressions – https://www.w3schools.com/java/java_strings.asp
2. W3Schools – <https://docs.oracle.com/javase/tutorial/essential/regex/>
3. Oracle Tutorials: Manipulating Characters in a String – <https://docs.oracle.com/javase/tutorial/java/data/manipstrings.html>
4. Oracle Tutorials: Basic I/O – <https://docs.oracle.com/javase/tutorial/essential/io/>

Data Structure and Integration in Program

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Various data structures.
- Difference between primitive and non-primitive data structures.
- Difference between tree and graph.
- Various tree data structures.
- Binary tree and how it is useful.
- Best data structure for your needs.

17.1 | Introduction

Data structures are essentially arrangements that can be used for storage and manipulation of the internal data of a computer program. What is interesting to note here is that most novice developers and programmers of Java programming language start using data structures long before they even know what they really are or how they are different from other data types.

While there are a lot of different data structures, some of the most commonly used ones include stacks linked lists and arrays. Having a strong understanding of the different types of data structures available in Java language and how they differ from each other is imperative for a number of reasons. The choice of data structure does not only affect the time taken by the application to perform crucial tasks; however, the choice of data structure has a strong connection with the effort that is required for implementation and the performance of the block of code, program, or application.

Since you now understand why it is important to know the difference between each of the data structures that are available, we will discuss a number of different data structures in great detail, take a look at their advantages and disadvantages, and explain how each of them can be used in programs.

17.2 | Introduction to Data Structures



Data structures are closely related to abstract data types (ADTs), which is another extremely important area in Java programming language. Best termed as a unique mathematical model for data types, an ADT is essentially defined by the user based on how it is expected to act in the program, block of code, or application where it is to be used, and the operations that are likely to be performed on it.

While data structures are essentially based on ADTs, the difference lies in the fact that there is a concrete implementation mechanism in place for data structures, and they cannot be arbitrarily used like ADTs.

As complex and beneficial as they are for efficient programming that gives optimal results, data structures can be classified and categorized into a number of different groups, each of which has its own characteristics, features, and properties. The focus of this chapter will be exclusively on data structures, their types, and their correct usage.



How do you program without any data structure?



17.3 | Classification of Data Structures

As mentioned, there are a number of different ways in which data structures of Java language can be classified. The most common classifications are primitive data structures and non-primitive data structures. Figure 17.1 shows the data structure hierarchy.

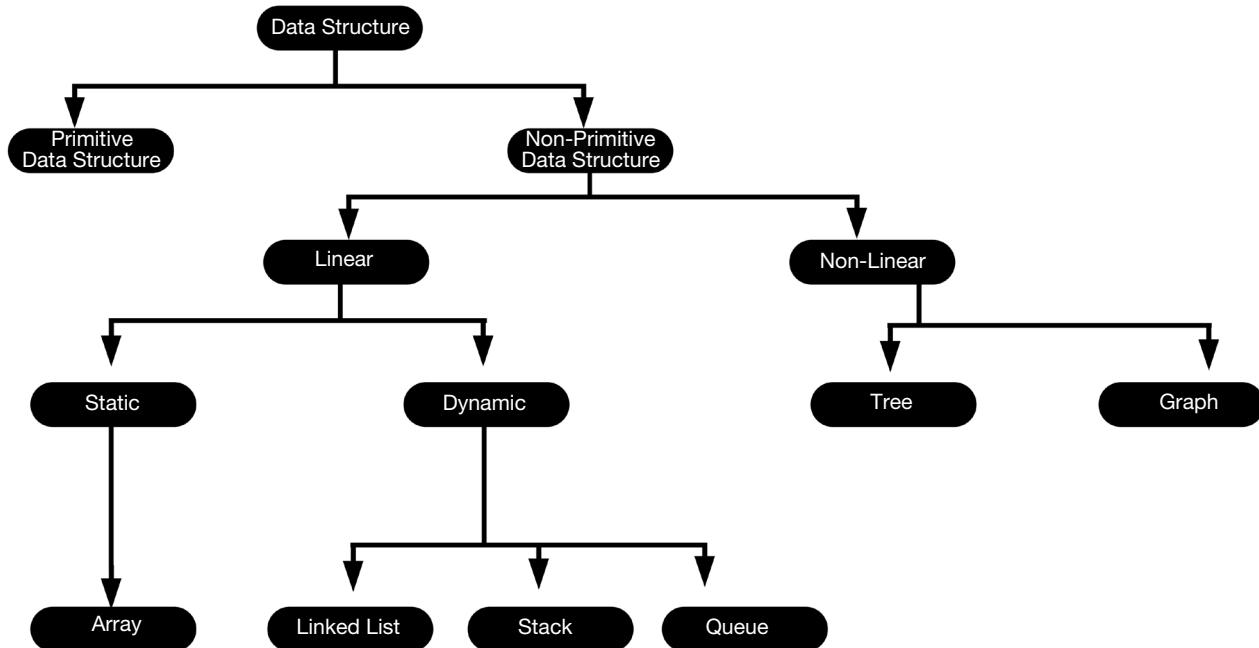


Figure 17.1 Data structure hierarchy.

17.3.1 Primitive Data Structures

Primitive data structures are nothing but the basic data types that are available in every programming language. They are used for the most fundamental of reasons. These predefined ways of storing data in Java language come with a limited set of operations that can be performed on them, essentially setting a limit to the number and types of situations for which they can prove to be beneficial.

We have learned about this in the Chapter 11, but let us revise it again. There are eight different primitive data structures available in Java language, namely, int, char, float, Boolean, long, double, short, and byte. Often referred to as the fundamental building blocks for the manipulation of data in Java, these data structures can be used to store the simplest types of values – that too of a single kind. Since primitive data types are very limited in what they offer, it is no surprise that there are certain situations where primitive data structures alone do not suffice. And that is where derived data structures come into the picture.

Derived data structures and user-defined data structures not only offer a much wider range of applications, but their benefits are also unique in a number of ways. We will learn about non-primitive data structures and how they differ from their primitive counterparts in the following sections. For now, we will define each of the primitive data structure of Java language along with examples of how primitive data structures can be declared and used in any program, application, or block of code, and what they can be used to achieve.

- Byte:** The Byte primitive is one of the basic predefined data structures of Java language, which is found in the form of an 8-bit signed two's complement integer. The Byte primitive has a maximum value of 127, and can take values as low as -128. The default value of this data structure is set as 0.

The fact that the Byte primitive requires far less storage space than the int type is one of the major reasons why it is so popular among developers. A Byte is four times smaller than an integer, which is why it can be easily used in certain scenarios in place of integers, especially when storage space is a concern.

Here is an example of how a Byte primitive can be declared:

```
byte a = -37;
```

2. **Short:** Like Byte, the Short primitive is also a two's complement signed integer. The difference between the two, however, lies in the fact that the Short data structure has 16-bit values. What this means is that the range of values for Short is much wider than for Byte, making the maximum possible value for Short 32,767 and the minimum possible value -32,768.

Like the Byte primitive, the Short primitive also has a default value of 0, and is used especially in situations where storage is a concern. This is because the Short type takes two times less space than the integer type.

Here is an example of how the Short primitive can be declared to be used in any program written using Java language:

```
short s1 = 32000;
```

3. **Int:** A signed two's complement integer, the Int primitive allows a 32-bit value, making the range of the data structure much larger than both the Short and Byte primitives combined. Since the primitive allows 32-bit values, the maximum value that can be stored in an Int type is $2^{31} - 1$, or 2,147,483,647, whereas the minimum value that is allowed in the Int type is -2^{31} , or -2,147,483,648. The default value for the Int type is 0.

Even though the Int type takes up a lot more memory than the Short and Byte types, it is often the data structure of choice for integer values, unless memory or storage space is a concern.

Here is an example of how an Int type variable can be declared in Java language:

```
int a = 500000;
int b = -500000;
```

4. **Long:** The Long primitive of Java language is a signed two's complement integer that can have a 64-bit value. Needless to say, the range of values allowed by Long is far wider than all other primitives in Java, with possible values ranging from a minimum of -2^{63} , or -9,223,372,036,854,775,808, to a maximum of $2^{63} - 1$, or 9,223,372,036,854,775,807.

Since this primitive takes up more memory and requires more storage space than variables of the Int type, the Long primitive is only used for integer values that will not be possible with the Int primitive. Additionally, values of the Long primitive are easily differentiable from values of variables of other primitives because they are always terminated with L. Similar to the default values of other variables discussed above, the default value for the Long primitive is 0L.

Here is an example of how a variable of the Long type can be declared in Java language:

```
long a = 19823290832L;
```

5. **Float:** The Float data structure in Java language is an extremely interesting and useful one for many reasons. The Float primitive allows single precision 32-bit values, and is primarily used for saving memory in situations where arrays of floating point numbers need to be used. Like the values of the Long type, values of the Float primitive are also easily differentiable thanks to the "f" at the end of all values of the data type.

The default value of the Float primitive is 0.0f, and the data type is never used for situations in which precise values are needed.

Here is an example of how a variable of the float type can be declared in Java language:

```
float f1 = 2.5f;
```

- 6. Double:** As the name suggests, the Double primitive is a double precision data structure in Java language that supports 64-bit values, giving it a wider range of possible values than the float data structure. This primitive is generally used as the default choice of programmers when they need to deal with decimal values.

Like the Float data structure, the Double primitive should never be used in situations where accuracy and precise values are imperative for the integrity of the program or application to be maintained. Additionally, values of Double variables are also easily differentiable as they end with a “d”.

Here is an example of how a variable of the Double primitive can be declared in Java language:

```
double d1 = 234.5;
```

- 7. Boolean:** The Boolean primitive is used for representation of a single bit of data in Java language. Unlike the other primitives and the range of values that were allowed in them as discussed above, the Boolean primitive allows only two values – false and true.

The only area where this primitive can be used is for tracking whether a condition will be true or false. The default value for variables of the Boolean type is set as false.

Here is an example of how a variable of the Boolean primitive can be declared in Java language:

```
boolean var = true;
```

- 8. Char:** The Char primitive is used for representation of a single 16-bit Unicode character in Java language. The range of values allowed by the Char primitive is rather limited, with the minimum possible value as 0, or “\u0000”, and the maximum possible value 65,535, or “\uffff”.

The Char primitive is only used in situations where a single character needs to be stored.

Here is an example of how a variable of the Char primitive can be declared in Java language:

```
char c1 = 'b';
```

As evident from the explanation of each of the primitives and the range of values permissible by each of them, it goes without saying that another mechanism for the storage of variables and their manipulation was crucial. Primitives of Java language are not only limited in the operations that can be performed on them, but the range of values allowed by them also limits the possibilities.

And this is exactly why non-primitive data structures such as arrays, stacks, and linked lists exist.

While they are essentially made up of primitive data structures, non-primitive data structures provide a lot more room for operations, and leave users with more to play around.

In the following section, we will learn about the most common non-primitive data types and explain how each of them can be used, their advantages, and how they differ from each other.



What is the difference between Float and Double?

17.3.2 Non-Primitive Data Structures

While non-primitive data structures are based on their primitive counterparts, the range of functionality offered by non-primitive data structures and the benefits that they offer are greater than primitive data structures. As shown in Figure 17.2, non-primitive data structures can be classified into two major groups – linear non-primitive data structures

and non-linear non-primitive data structures. Table 17.1 lists the differences between linear and non-linear non-primitive data structures.

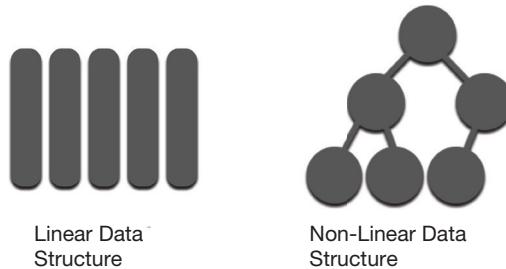


Figure 17.2 Representation of linear and non-linear data structures.

Table 17.1 Linear data structure vs non-linear data structure

Parameter	Linear Data Structure	Non-Linear Data Structure
Basic	Elements are arranged adjacent to each other	Elements are arranged in a sorted order
Traversing of the data	Data elements are traversed in one go	Data elements cannot be traversed in one go
Ease of implementation	Simple	Complex
Levels involved	Single	Multiple
Examples	Array, LinkedList, Queue, Stack, etc.	Graph, Tree
Memory utilization	Inefficient	Efficient

QUICK CHALLENGE

Define a situation in which you could use non-linear data structure.

17.3.2.1 Linear Non-Primitive Data Structures

For a data structure to be considered as linear, it is imperative for the elements that make it should form a linear list. Another requirement is that the constituent elements of the data structure should be connected to each other adjacently and follow a certain order. Additionally, for a data structure to be considered linear, the memory that it consumes should also be in linear fashion, and the storage of elements in the memory must be sequential.

Linear non-primitive are also differentiable from their non-linear counterparts because a certain amount of memory has to be declared in advance for the linear data structure, unlike in the case of non-linear data structures. While this requirement often results in loss of memory and mismanagement due to improper utilization of memory, doing so is imperative if you want to use linear data structures. Additionally, since memory and element storage for linear data structures works in a linear and sequential fashion, constituent elements of a linear structure can only be reached sequentially, and only a single element of the data structure can be visited without a reference. Since it is imperative for adjacency relationships to be maintained for data structures, the operations that can be performed on linear data structures are also limited in a number of ways. What this means is that it is not possible for one to insert or delete constituent elements of a linear data structure arbitrarily, and all operations must be performed in a sequential fashion. Here are some of the operations that can be performed on linear data structures in Java language:

1. Addition of elements.
2. Deletion of elements.
3. Traversing the data structure.

4. Sorting the constituent elements of the data structure.
5. Searching for a constituent of the linear data structure.

Now, we will look at a few of the most popular linear data structures in detail, and learn how they can be used in programs or applications.

17.3.2.1.1 Arrays

Arrays are perhaps the most popular data structure in Java and other programming languages. Often built into programming languages, arrays act as a great starting point to not only introduce novice programmers to the concept of data structures, but also to understand how object-oriented programming (OOP) and data structures go hand in hand. In this section, we will learn how to create arrays from scratch, use them effectively and efficiently, and perform operations on them. This will help users get an idea of how data structures really work; we will then move on to more complicated linear and non-linear data structures that are used in Java language.

Arrays in Java are created dynamically and can contain multiple elements that essentially define the size or length of the array. What this means is that if an array contains x elements, the length of the array will be x .

While all of the constituent elements of an array have the same name, they each have a unique reference based on where they lie in the linear sequence of the data structure. To reference any particular element in the data structure, you will need the index of the element in the array, which will always be in the form of a non-negative integer. Since the index of the first element is always 0, the last constituent element of any particular array will always be $x - 1$, where x is the length, size, or total number of elements present in the data structure.

It is important to note that even though it was mentioned that an array can contain multiple elements, the condition is that all of these elements must be of the same type, which is often called the component type of an array. What this means is that even if an integer array has been created, you will not be allowed to enter float values in the array. If an integer array needs to be created as in the case mentioned above, we will declare the integer array as follows:

```
int age[]; // This line of code declares an integer array to hold ages
```

It is also important to note that arrays in Java can have multiple dimensions, but that does not change the fact that there are no restrictions on the element type or component type of the array. Additionally, whenever a variable of the array type is created, it does not set aside any amount of memory or create the object for the array. Instead, the only thing that really happens is the creation of the variable for the array – a variable that may be used to contain the reference to the array itself.

Now that you have a basic understanding of how arrays work, their characteristics, and the operations that can be performed on them, we will focus on the practical implementation of arrays and how that can be done in Java.

Declaration of simple arrays: An array in Java language is essentially a list of elements that have the same type and are referred to by the same name. As seen above, there is not much that needs to be done to declare an array in Java besides adding a pair of square brackets after the name of the array that you wish to create.

Another way to do this is by adding the pair of square brackets after the type of array, as follows:

```
float[ ] temperature;
```

Therefore, the permissible syntax for the declaration of an array is as follows:

```
type[ ] name;
```

or

```
type name[ ];
```

As mentioned above, simply declaring an array does not suffice. Instead, you will have to create or initialize your array for you to be able to effectively use it in the program, application, or block of code. Next, we will explain how an array can be created, and how memory can be allocated to ensure that the array that was declared is functional and can be used in the program.

Creating a one-dimensional array: Whenever an array is to be created to be used in Java, the **new** keyword will be used, and the length or size of the array will be specified in square brackets. Here is an example of how this will work:

```
Int age[ ]; // this line of code declares a one-dimensional array of the integer type named age
age = new int[4]; // this line of code creates a new array after declaring it
```

Initialization of a one-dimensional array: This is perhaps the easiest and simplest part of the process. The only thing that needs to be done for initialization of an array in Java is to place values in curly braces separated by commas. Here is how this can be done:

```
int age[4] = {11,22,35,64};
```

Now that you understand how an array can be declared, created, and initialized in Java, here is a sample code that will help put things into perspective:

```
package java11.fundamentals.chapter17;
public class ArrayExample {
    public static void main(String args[]) {
        // Following code declares an int array
        int myIntArr[] = { 1, 5, 993, 35 };
        // Following code iterates through array elements and print them one by one
        for (int b = 0; b < myIntArr.length; b++) {
            System.out.println(myIntArr[b]);
        }
    }
}
```

The above program produces the following result.

1
5
993
35

As seen above, the declaration, creation, and initialization of arrays in Java language is extremely easy and simple. However, there are advantages and disadvantages to the concept of arrays in Java language.

Although arrays make for a great data structure and provide a resourceful mechanism for the storage and manipulation of large amounts of data, the major problem is that they can only be used to store data elements of a single type. Moreover, once an array has been declared, it is not possible for it to be changed in any way. What this means is that once you have declared an array, it will not be possible to increase or decrease its size, resulting in memory wastage.

**QUICK
CHALLENGE**

Define two arrays of same size and integer type. Write a program to create a third array which contains the values which are the result of multiplication of each element from the first array and second array. [The first element of first array will get multiplied with the first element of second array, the second element of first array will get multiplied with the second element of second array, and so forth.]

17.3.2.1.2 Lists and Queue

The List interface in Java language is particularly interesting for a number of reasons. While there are several implementations of the List interface, they all work on an identical principle. Much like arrays, elements in Lists of Java can also be accessed according to their index or position in the list. The same mechanism is also used for the addition of elements, and duplication is possible within lists as well. Additionally, since List extends Collection in Java language, all of the operations of Collection are supported by List as well.

Queue is another data structure which works on the First In, First Out (FIFO) principal. We have covered these topics in detail in Chapter 13.

17.3.2.2 Non-Linear Non-Primitive Data Structures

The following data structures come under the non-linear non-primitive data structures category. Data in this structure is not arranged sequentially but in a sorted order. In this type of structure, all the operations are done in a non-sequential manner such as traversal of data elements, insertion, and deletion.

These types of non-linear data structures are memory efficient, as they use memory resourcefully and do not need pre-memory allocation. The two types of data structures available are *tree* and *graph*. Data is arranged in a hierarchical relationship in the tree structure, which involves the relationship between the child, parent, and grandparent.

17.3.2.2.1 Tree Data Structure

As shown in Figure 17.3, the following are tree related data structures:

1. General tree.
2. Forests.
3. Binary tree.
4. Binary search tree.
5. Expression tree.
6. Tournament tree.

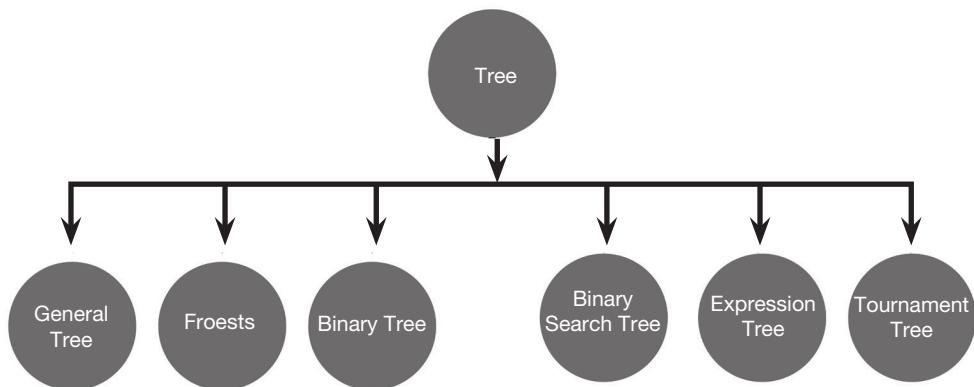


Figure 17.3 Tree data structures.

Most of the time, you will be using binary tree and binary search tree data structures. These are two of the popular data structures commonly used to solve many problems. Figure 17.4 shows the tree representation in Java.

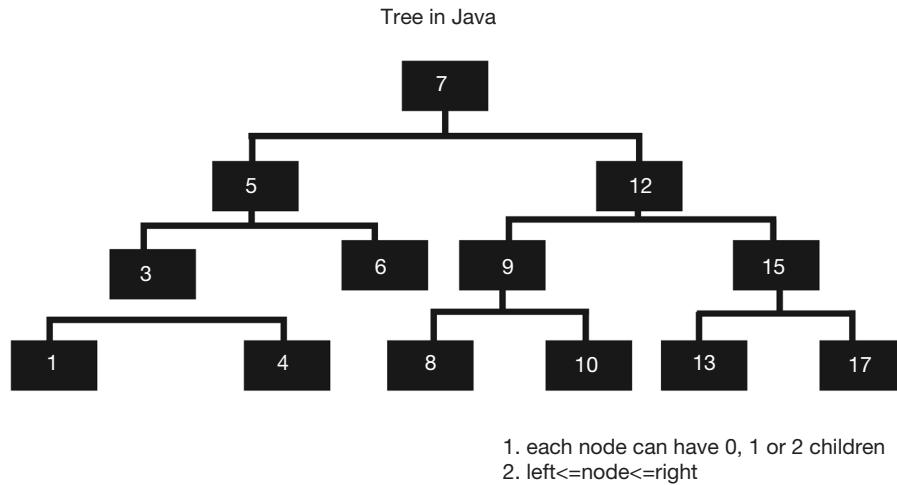


Figure 17.4 Tree representation.

1. **General trees:** Many times we encounter a set of data which is not in a format that can be handled by binary tree. Binary tree can have maximum 2 nodes. Hence, cases like displaying organization hierarchy cannot be handled by binary tree as the CEO (root node) can have more than 2 vice presidents under it. For this type of data, we need a tree data structure which can handle multiple nodes under any root node. This data structure is known as general tree, which is shown in Figure 17.5.

A general tree T is a tree which has one root node r and finite set of one or more nodes. Tree T starts with root node r . If the first set ($T \setminus \{r\}$) is not empty, then the rest of the nodes are divided into $n \geq 0$ disjoint subset trees like T_1, T_2, \dots, T_n which are called *subtrees*. Each of these trees has a root node which looks like r_1, r_2, \dots, r_n , respectively. These root nodes are children of r .

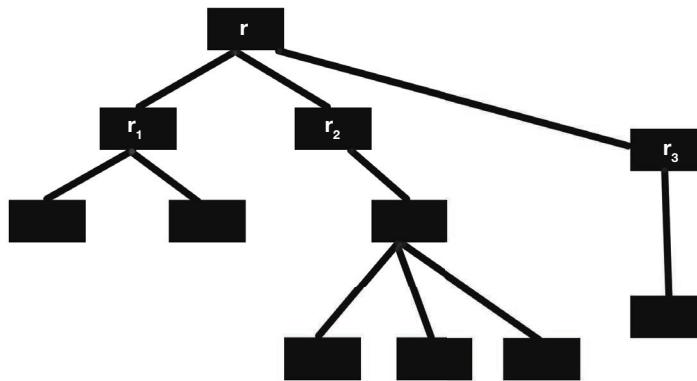


Figure 17.5 General tree structure.

QUICK CHALLENGE

Give an example of a real-life scenario which is suitable to be represented as a general tree.

2. **Forests:** An arbitrary set of trees is called Forest. Figure 17.6 shows representation of Forests. A Forest can start with one root node; it then grows in an ordered fashion, with one node having any number of child nodes. The children of a node are sequenced as first, second, and so on. Contrary to the binary tree, this tree does not have a concept of left and right. However, we can sequentially draw this tree from left to right. An Ordered Forest is nothing but an ordered set of ordered trees, which are also termed as Orchard.

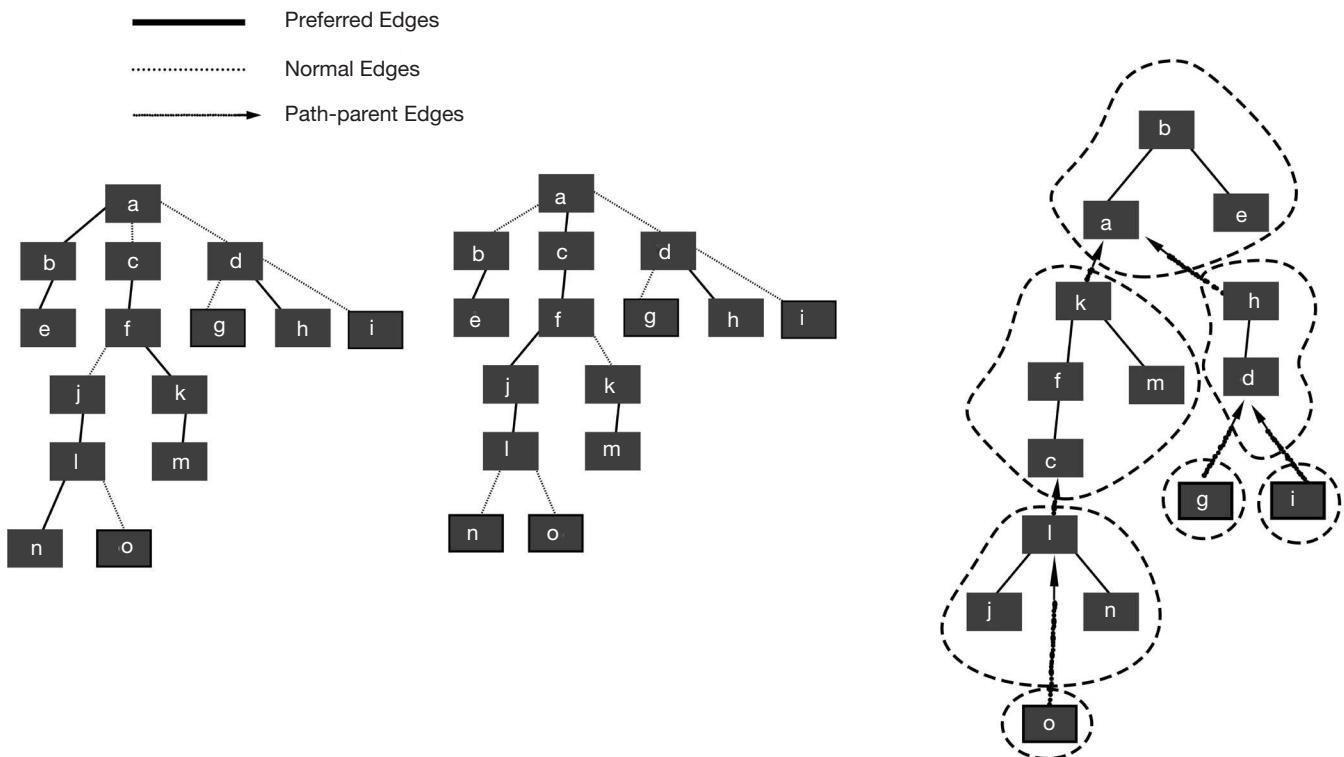


Figure 17.6 Representation of Forests.

3. Binary tree: As we already know, tree data structure is hierarchical in nature. Binary Tree is a subset of Tree in which each node has at most two children. Figure 17.7 shows Tree vs Binary Tree. These children are referred as the left and right child. This tree is implemented using links. The topmost node is the pointer in the tree. If tree is empty, then the value of root is NULL.

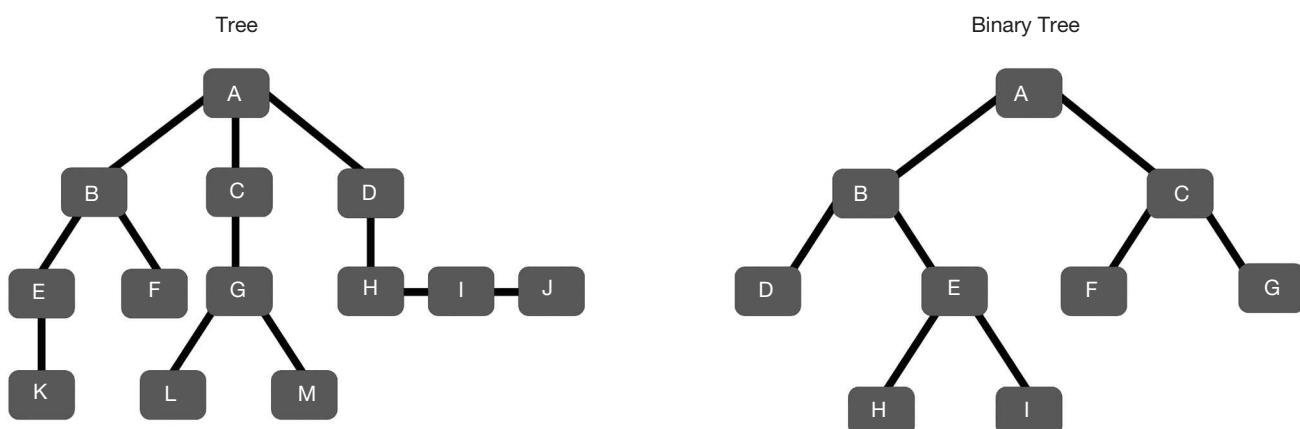


Figure 17.7 Representation of Tree vs Binary Tree.

A node has three parts: (a) data, (b) pointer to left child, and (c) pointer to right child.

There are two ways in which a binary tree can be traversed:

- (a) **Depth-first traversal:** In this case, there are three ways to traverse:
 - In the first way, it starts with left node, then root node, and then right node is accessed. This is called *inorder (Left-Root-Right) traversal*.
 - In the second way, it starts with the root node, then moves to the left node, and then comes to the right node. This type of traversal is called *preorder (Root-Left-Right) traversal*.
 - The third way is to start with left node, then move to the right node, and finally come to the root node. This is called *postorder (Left-Right-Root) traversal*.
- (b) **Breadth-first traversal:** In this case, the traversal takes place level-by-level. As name suggests, the focus is on the breadth of the level. In other words, at every level, the traversal takes place on the full width of the tree before going to the next level. Let us take an example of the binary tree shown in Figure 17.7. In this example, node A is at level 1, nodes B and C at level 2, nodes D, E, F, and G are at level 3, and nodes H and I are at level 4. In this case, the algorithm will first traverse through node A, then through nodes B and C, then nodes D, E, F, G, and finally it will traverse through nodes H and I.

Following are some of the properties of binary tree:

- (a) The maximum number of nodes at level x of a binary tree is always 2^{x-1} .
- (b) The maximum number of nodes of height y is 2^{y-1} .
- (c) The maximum number of levels or height in tree with n nodes is $\log_2(n+1)$.
- (d) Every node in the tree has 0 or 2 children.
- (e) Total number of nodes with 2 children are always less by one than the number of leaf nodes (i.e., $L = X + 1$, where L is number of leaf nodes and X is internal nodes with two children).



Construct a binary tree for the following array:

```
myArray[] = {1, 2, 3, 4, 5, 6}
```

4. **Binary search tree:** This tree is similar to the Binary Tree. It is mainly used for faster search over LinkedList, but it is slower than arrays. For insertion and deletion, it is better than arrays but not LinkedList. This is an important data structure that you should be aware of. It provides efficient search with $O(\log n)$ complexity.



Can a Binary Search Tree's topmost node contain a null node?

It has three more other properties than Binary Tree.

- (a) It contains keys less than the node's key on the left side.
- (b) It contains keys greater than the node's key on the right side.
- (c) Both sides, left and right, must be a Binary Search Tree.

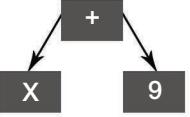
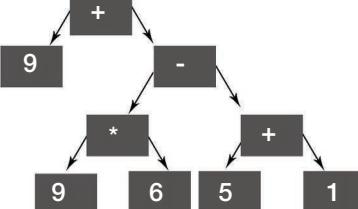
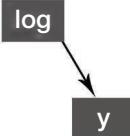
A Binary Search Tree is quite useful in applications such as e-commerce, where products are getting added or deleted from the inventory constantly and then presented in a sorted order.



Can a Binary Search Tree's topmost node contain more than two elements?

5. **Expression tree:** As name suggests, an Expression Tree is one made up of expression, wherein each internal node contains an operator and each leaf node contains an operand. For example, in the expression $x + 9$, x and 9 are operands and $+$ is the operator. Hence, the internal node will start with $+$, and on the left we will have x and on right we will have 9 . Table 17.2 shows a few examples of Expression Trees.

Table 17.2 Examples of Expression Trees

Expression	Expression Tree	Inorder Traversal Result
$(x+9)$		$x+9$
$9+(9*6-(5+1))$		$9+9*6-5+1$
$\log(y)$		$\log(y)$

6. **Tournament trees:** A complete Binary Tree with n external nodes and $n - 1$ internal nodes is called Tournament Tree. All the external nodes are characterized as players and all the internal nodes are characterized as winners. Winner node is situated between the two external nodes termed as players. As we have just discussed, a Tournament Tree gets $n - 1$ internal nodes and n external nodes. Thus, to find the winner, we need to eliminate $n - 1$ players. In simple term comparisons, this means that we need a minimum of $n - 1$ games.

There are two types of Tournament Trees:

1. **Winner tree:** A Winner Tree can be defined as the complete Binary Tree, where every node characterizes the smaller or greater of its two children. As in the case of the Binary Tree, it starts with the root, so root becomes the representation of the smaller or greater node. The Tournament Tree winner can be found by looking at the smallest or greatest n key in all the sequences. The time complexity of this tree is $O(\log n)$.
2. **Loser tree:** A Loser Tree can be defined as the complete Binary Tree with n external nodes and $n - 1$ internal nodes. The loser is stored in the internal nodes of the tree. The root node at [0] is the winner of the tournament. The corresponding node is the loser node.

Uses of tournament tree: The following are the uses of this type of tree:

1. Finding the smallest and largest element in the array.
2. Sorting.
3. Merging M-way. (M-way merge is like merging M sorted arrays to get single sorted array).



What are the benefits of using a Tournament Tree?

17.3.2.3 Graph Structure

Graph data structure stores connected data where nodes are connected to each other in a network fashion. Think of a social media platform. You are connected to your friend, your friend connects back to you, there is someone connected to your friend, there may be someone whom you also know connected to your friend, and so on and so forth. This is how graph is setup. A

graph contains vertices and edges. The vertex represents an entity like people and the edge is the relationship between those entities.



Can a graph be useful to store a dictionary values?

Figure 17.8 shows a graph, edges, and vertices.

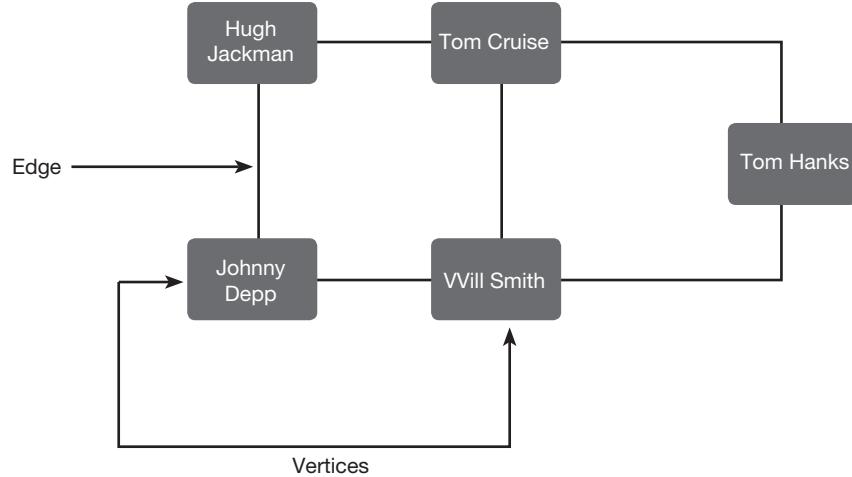


Figure 17.8 Example of a graph.

In the above example, you can see that the nodes are named as Johnny Depp, Will Smith, Hugh Jackman, Tom Cruise, and Tom Hanks. These are entities and the connection between these nodes are edges.

There are a few variations to the graph data structure of Figure 17.8 . These are discussed in the following subsections.

17.3.2.3.1 Directed Graph

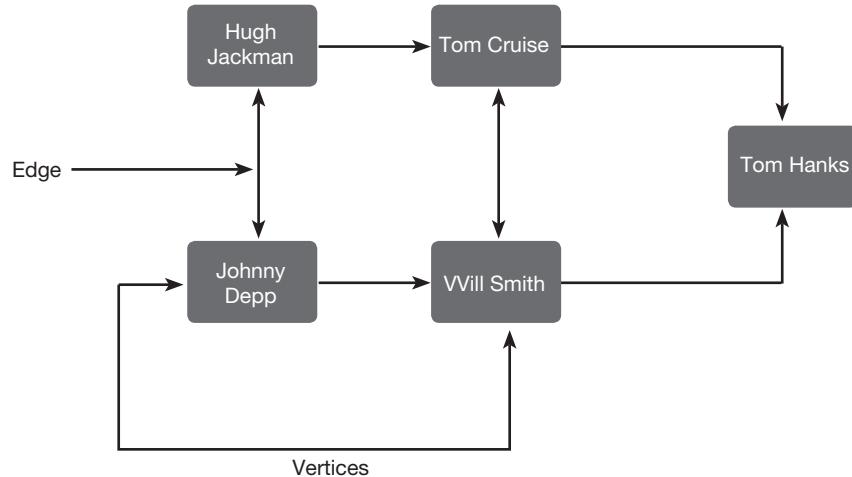


Figure 17.9 Example of directed graph.

In the above example shown in Figure 17.8, we do not see any direction. That is, the nodes are just connected and do not carry any specific direction. If we add direction to the edges, then the graph becomes a directed graph. It means that the edges

point the relationship direction between nodes. In the social media example, this could be used to show who sends a friendship request. These edges can carry bidirectional relationship as well as shown in 17.9. Bidirectional relationship exists between two nodes which are connected both ways. In other words, the connection between node A and node B is taking place from node A to node B and vice versa.

17.3.2.3.2 Weighted Graph

We have now seen a graph with edges and one with direction-based edges. If these edges carry relative weight, they become a weighted graph as shown in Figure 17.10. In the social media example, each connection carries direction, like the connection between you and your friend A. If we add the number of years you know your friend to the edge, then that becomes the weight of the relationship.

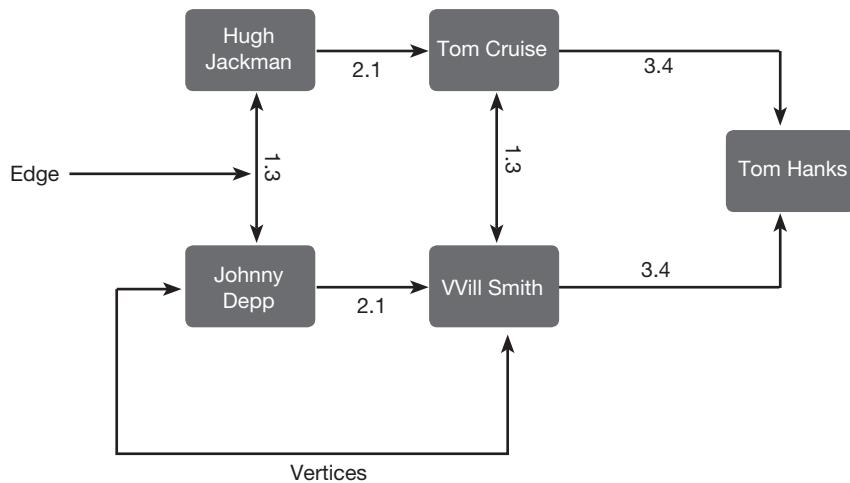


Figure 17.10 Example of weighted graph.

17.3.2.4 Graph Representations

A graph has two commonly used representations – adjacency matrix and adjacency list. It also has other representations that are less commonly used – incidence matrix and incidence list.

17.3.2.4.1 Adjacency Matrix

An adjacency matrix is a square matrix. In simple terms, as shown in Figure 17.11, an adjacency matrix shows if the elements in the graph are next to each other. The number of vertices in the graph defines the dimensions of the graph. This matrix has simple representation, which contains values of 0 or 1. A value of 1 indicates that there is an adjacency between row and column. And a value of 0 indicates that there is no adjacency between row and column. Let us take our example and see how it can be represented as a graph.



Can an adjacency matrix be useful to define edge weight?

This is the easiest representation to implement and follow. The complexity of edge removal is $O(1)$ time and the edge find between two vertices can also be done in $O(1)$ time. However, the adjacency matrix takes a larger space of $O(V^2)$ time (here, V is vertex set) even though it may contain lesser number of edges. The edge addition takes $O(V^2)$ time.

	Hugh Jackman	Tom Cruise	Tom Hanks	Johnny Depp	Will Smith
Hugh Jackman	0	1	0	1	0
Tom Cruise	1	0	1	0	0
Tom Hanks	0	1	0	0	1
Johnny Depp	1	0	0	0	1
Will Smith	0	1	1	1	0

Figure 17.11 Example of adjacency matrix.

17.3.2.4.2 Adjacency List

As shown in Figure 17.12, an adjacency list is simply an array of lists. The number of vertices in the graph determines the size of the array. Array[i] represents the i th vertex's adjacent vertices. This list can also represent a weighted graph. The weights of edges in the weighted graphs are represented as lists of pairs. The adjacency list representation takes less space $O(|V|+|E|)$, where V is vertex set and E is edges. In a worst-case scenario, it may take $O(V^2)$ space, even though there may be $C(V, 2)$ number of edges. Adding a vertex is relatively easier than in an adjacency matrix. However, the edge find process between two nodes may be expensive, such as $O(V)$ times.



Figure 17.12 Example of adjacency list.

17.3.2.4.3 Incidence Matrix

Incidence matrix is useful to show the relationship between objects of two classes. This matrix is built in a way that it contains one row for each element of first class and one column for each element of second class. If first class row x and second class row y are related, it contains 1 in row x and column y which is called incident in this context and it contains 0 if they are not related.

Incidence matrices are frequently used in graph structure, such as undirected and directed graphs.

- Undirected graph:** Let us take the example shown in Figure 17.13 of an undirected graph, X . This graph contains vertices and edges; say, n number of vertices and m number of edges. So, this matrix Y of graph X is of $n \times m$ matrix. According to the incidence matrix case, this matrix will have $Y_{i,j} = 1$ if the vertex v_i and edge e_j are related also called incident and 0 if they are not related.

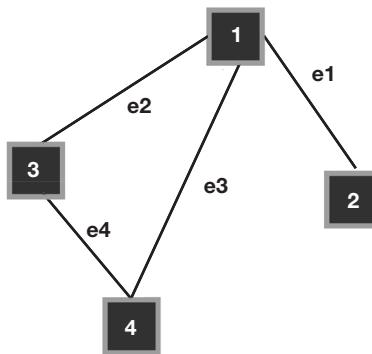


Figure 17.13 Undirected graph.

Table 17.3 shows the matrix based on the undirected graph shown in Figure 17.13.

Table 17.3 Incidence matrix for undirected graph

Sl. No.	e1	e2	e3	e4
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

- Directed graph:** As the name suggests, directed graph is one which contains vertices connected by edges. These edges carry a direction with them. The incidence matrix of a directed graph, D , is shown in Figure 17.14. It contains n number of vertices and m number of edges of matrix M . So, matrix M is an $n \times m$ matrix, as shown in Table 17.4. This matrix content is set based on the directed relationship. For example, if edge e_j leaves vertex v_i then position $M_{i,j} = -1$, if it enters vertex v_i then 1 and 0 if it does not enter.

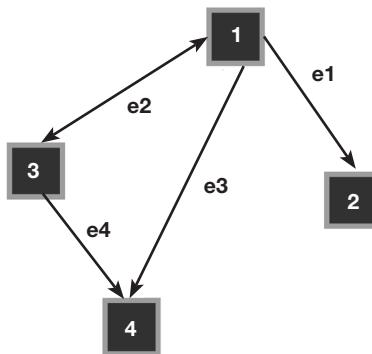


Figure 17.14 Directed graph.

Table 17.4 shows the matrix based on the directed graph shown in Figure 17.14.

Table 17.4 Incidence matrix for directed graph

Sl. No.	e1	e2	e3	e4
1	-1	1	-1	0
2	1	0	0	0
3	0	1	0	-1
4	0	0	1	1

Summary

Data structure is an important concept of programming. In any programming language, you will encounter at least one data structure that you will need to complete your program. There are various types of data structures. Some are very common, while others are rarely used but very useful in solving complex problems.

In this chapter, we have learned the following:

1. What is a data structure?
2. How to use data structure in programs.
3. Linear and non-linear data structures.
4. Benefits of using Binary Tree.
5. Difference between tree and graph.
6. Directed and weighted graph.
7. Adjacency matrix and adjacency list.

In Chapter 18, we will learn about lambdas and functional programming. We will spend time understanding lambdas and explore the ways to use it in a program. We will explore the use of functional programming.

Multiple-Choice Questions

1. Which of the following has elements arranged to each other?
 - (a) Linear Data Structure
 - (b) Non-Linear Data Structure
 - (c) Both (a) and (b)
 - (d) None of the above
2. Which one of the following data types is utilized only for positive values in data structure?
 - (a) Arrays
 - (b) Unsigned
 - (c) Signed
 - (d) Boolean
3. What is the minimum number of nodes that a binary tree can possess?
 - (a) Two
4. You can access an array by referring to the indexed element within the array.
 - (a) True
 - (b) False
5. Which of the following trees has each internal node containing an operator and each leaf node containing an operand?
 - (a) Winner Tree
 - (b) Loser Tree
 - (c) Expression Tree
 - (d) Binary Search Tree

Review Questions

1. What is data structure?
2. How do we use data structure?
3. What is linear data structure?
4. What is non-linear data structure?
5. What is the difference between linear and non-linear data structure?

6. Give at least two scenarios on when to use linear and when to use non-linear data structure.
7. Define tree data structure.
8. What are the frequently used subsets of tree data structure?
9. What is the search complexity of Binary Search Tree?
10. What is Expression Tree?
11. What is graph structure?
12. Give one real-life example of graph structure in use.
13. What is directed graph?
14. What is weighted graph?
15. How do you create adjacency matrix?
16. How do you create adjacency list?
17. Which graph representation is faster in finding the edge between two nodes?
18. What is the complexity in removing edge in adjacency matrix?

Exercises

1. Plot your Facebook account data and figure out your personal graph like your friends, their friends, common friends, etc. Design a graph and add weight to it.
2. Create an adjacency matrix and adjacency list for the graph in Question 1.
3. Create a chart to show the differences, advantages, and disadvantages between linear and non-linear data structure.

Project Idea

Design your own social media platform. Create pages to add users and let them search the platform to look for connections. Write a detailed plan on how to store and retrieve data. Figure out the best possible data structure for

this problem and design one. Collect as much as data and run various algorithms on it to learn how you can extend functionalities for your social network.

Recommended Readings

1. Allen B. Downey. 2017. *Think Data Structures: Algorithms and Information Retrieval in Java*. O'Reilly Media: Massachusetts
2. James Cutajar. 2018. *Beginning Java Data Structures and Algorithms: Sharpen your problem solving skills by learning core computer science concepts in a pain-free manner*. Packt: Birmingham
3. Mr Kotiyana. 2018. *Introduction to Data Structures and Algorithms in Java*. [Independently Published]
4. Suman Saha and Shailendra Shukla. 2019. *Advanced Data Structures: Theory and Applications*. Chapman and Hall/CRC: London

Lambdas and Functional Programming

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Functional programming.
- Higher-order and first-class functions.
- Pure functions and their characteristics.
- Recursion functions, evaluation, and referential transparency.
- Working examples of all functions using Java.
- Lambdas.

18.1 | Introduction

Programming applications in different languages require a specific style known as programming paradigm. In the earlier days, programmers used to code in C via procedural programming. Procedural programming is also called *imperative programming*. In a procedural paradigm, the code runs step-by-step, similar to kitchen recipes. The paradigm was excellent for its time, but as the complexities in the developing world grew, it was realized that there was a need for a reusable code that could make programming easier for all the stages of software lifecycle.

Companies revolutionized their business processes by adding a desktop or web system to achieve greater business productivity. Programmers soon realized that procedural programming was ineffective and cumbersome for their requirements.

As a result, object-oriented paradigm was introduced. In object-oriented programming (OOP), “objects” were introduced that mirrored the real world where each object can have its distinct behavior and state. Thus, developers began viewing code with respect to the real world and added functionalities accordingly.

OOP was globally instrumental in shrinking the size of codebases. As users were able to assign behavior and states to each object, they used OOP components such as inheritance, polymorphism, encapsulation, and abstract to achieve unforeseen levels of productivity. The programs that once required 5,000 lines of code were now reduced to only 500 lines.

OOP manages to resolve several issues, but with advancements in academic as well as industry landscapes, it was simply not enough to work in all types of environments. Thus, a newer programming paradigm achieved success and became popular for programming. This new paradigm, called *functional programming*, is supported by mainstream languages such as Python, PHP, and Java, while several languages such as Haskell are gaining prominence due to their effectiveness for functional-based applications.

18.2 | Functional Programming

As the name suggests, functional programming relies on “functions”. Here, functions can refer to any operation or feature that has to be added into the application. With functional programming, there is no requirement for modifying the other components of the code in the process of adding a new function.

These functions can be seen as similar to real-world formulas in mathematics. In fact, functional programming is heavily based on mathematical functions. In mathematics, a problem is tackled by reasoning and solving with a technique that the real theory behind all the complex steps of the function are abstracted, and the problem solver can focus more on higher-level complexities by designing a solution. Likewise, functional programming can be seen as providing a greater level of abstraction in software development.

Functional programming prevents modification in states and mutability of data. Instead of statements, expressions are used in functional programming.





How is functional programming different from object-oriented programming?

18.2.1 Fundamental Concepts of Functional Programming

Before writing functional paradigm-based code, it is necessary to familiarize ourselves with the following concepts for the production of quality code. Without properly interpreting the underlying theory behind functional programming, our implementation can suffer in the production of powerful code.

18.2.1.1 Higher-Order and First-Class Functions

A higher-order function exists in both mathematical and computer science worlds. Generally, a higher-order function can be described as possessing any of the two properties.

1. The result type of the function is also a function. For instance:

```
// Here A is a function that takes an integer value of total
function A ( int total)
{
    // The function performs some processing and adds it to return another function
    return marks();
}
```

2. The function's arguments include a single function or multiple functions. For example, we have defined a function `areaofRectangle`. The function takes the argument in the form of another function `values` that saves length and breadth of the rectangle and passes it to the former function.

```
// Here areaofRectangle is a function that has taken another function values for the
calculation of area
public areaofRectangle (values)
{
    // Since values is returned so the function can be considered as a higher-order
    function.
    return values;
}
```

Other than higher-order functions, we also have to acquaint ourselves with first-class functions. First-class functions are similar to higher-order functions. However, in first-class function, it is mandatory to follow both the conditions of a higher-order function; that is, a first-class function must return another function and contain a function for its parameter arguments also. Thus, each first-class function can be termed as higher-order function.

However, there is a thin line that separates both types of functions. When we use the term “higher order”, our focus is more inclined towards the mathematical aspect of the problem, while “first class” is used to view the problem in a computer science oriented approach.

18.2.1.2 Pure Functions

As already discussed, one of the core components of a functional application is the presence of expressions. These expressions are also called *pure functions*. They are beneficial because they do not incur any significant side effects on input/output operations or memory. In functional programming, a function has a side effect when it is able to change states of data that lie outside its scope. A pure function has the following characteristics:

1. Sometimes, an expression is useful for a while. However, with the passage of time newer changes to the application may render the expression as useless. In other paradigms where statements are used in place of expressions, removing these statements may have an adverse impact on the application. However, in functional programming by eliminating expressions from an application, there is no effect on the operation of any other expression.

2. In computer science, sometimes a certain strategy called *memoisation* is used for increasing the speed of computer programs. In memoisation, the results of a function are stored in the cache so they can be returned when the function is called using the same inputs. Pure functions support memoisation; that is, if a pure function uses the same inputs as arguments and is continuously called by the applications, the system resources can be saved by issuing the same result through caching.
3. When side effects are not supported by the complete language, then a compiler can process expressions by any evaluation technique.
4. Pure functions can employ parallelism. What this means is that if the data that is stored by two pure expressions is separate from any relationship (logical or data dependency), then both expressions can be simultaneously processed. Likewise, they can also be rearranged.

QUICK CHALLENGE

Give an example of pure function.

18.2.1.3 Recursion

In procedural and objective paradigms, looping (iteration) of elements is performed with the use of loops (for loop, while loop). On the other hand, functional languages are geared towards the use of recursion for looping.

Recursion is a programming concept in which a function calls (invokes) itself. A recursive function contains conditions that repeat calling the function until it reaches to a base case. Some people believe that using loops or recursion is a matter of preference. However, recursion is productive in many cases where it:

1. Reduces the lines of code.
2. Reduces the possibility of errors.
3. Reduces the cost of the function.

Recursion techniques can be used by higher-order functions by utilizing anamorphisms and catamorphisms. As a result, the higher-order functions contribute in the development of control structures (like loops) in procedural programming languages.

Usually, functional languages support unrestricted recursions and are also equipped with Turing complete. Thus, the halting problem is undecidable; that is, an issue cannot be responded with a yes or no conclusion. A halting problem is one in which there is an issue of determining whether a computer program (containing an input) has to be stopped or allowed to run for an infinite period of time.

Recursion in functional programming paradigms also need some sort of “inconsistency” that is added into the logic of the language’s *type system*. (The set of rules that adds the type property to a construct in a programming language is called type system.) A few functional languages limit recursion types and may allow only the use of well-founded recursion.

QUICK CHALLENGE

Define Anamorphisms and Catamorphisms.

18.2.1.4 Evaluation

A function language can be grouped by two factors – strict and non-strict. These factors are influenced by the language’s ability to evaluate an expression with the arguments of the function. To understand strict evaluation and non-strict evaluation, let us take the following example where we have to print length of 5 elements:

```
print length([2-1, 5+7, 1/0, 8*8, 4+9])
```

If this expression gets evaluated by a non-strict evaluation, then a length of 5 is returned (as there are 5 elements in the list). Non-strict evaluation does not attempt to go into the intricacies of the elements. On the other hand, if strict evaluation is used, then the expression does not succeed because it finds an error in the third element of the list (1/0). Thus, strict evaluation delves into the details of expression to determine the logic and semantics of the elements. However, when the elements of a function are required for evaluation due to some calling, then non-strict evaluation assesses them in the same manner as strict evaluation.

Non-strict evaluation is implemented using graph reduction. Languages such as Haskell and Clean adopt non-strict evaluation.

18.2.1.5 Referential Transparency

Assignments statements are not supported in a functional program. What this means is that when a variable is assigned and defined with a value, then it cannot be modified in the future. Due to this property, the side effects of an application are reduced to a significant extent as the variables' values are changeable in the duration of execution processes. Hence, functional programs can be considered as referentially transparent.

Suppose there is a basic program in C that has the following expression, where the value of a keeps getting changed because of successive evaluations:

```
a = a * 20
```

Let us consider the initial value of a as 1. Now, after a single evaluation the value of a becomes 20. An additional evaluation can increase the value of a to 400. Since using any of these values changes the program's semantics, the expression cannot be said as referentially transparent. This is because its actual value is being continuously changed implicitly. In functional programming, the above example can be changed into the following function:

```
int addone(int a) {return a+2;}
```

Now, this evaluation always returns a fixed answer as the value of a is not modified implicitly. Likewise, the function does not incur with any side effect. Hence, it can be said that functional programs are inherently referentially transparent.

QUICK CHALLENGE

Give an example of referential transparency.

18.3 | Functional Programming in Java

Due to the growing demand of the functional paradigm in development circles, the release of Java 8 and Java 9 came with extensive support for functional programming. While functional programming was still possible in earlier versions of Java, there were too many restrictions to take Java seriously. However, with Java 8 and the recent Java 9 release, Oracle has transformed Java as a formidable option for functional enthusiasts. Let us visit some of the functional aspects in Java.



What are the other languages that support functional programming? What advantage does Java have over them?

18.3.1 Pure Function

Earlier we talked about pure function as a general concept. In Java 9, a pure function can be written as:

```
public class PureFnObject{
// the values of x and y are taken as input arguments
    public int subtract(int x, int y) {
        // The function returns the subtracted answer
        return x - y;
    }
}
```

There are two factors that make this function a pure one. First, you can observe that the subtract function is entirely reliant on the input arguments. Second, there are no side effects of the function because the values of x and y can be only changed inside the function. On the other hand, the same example can be written in the following code where the function is non-pure.

```
public class NonpureFnObject{
    private int x = 0;
    public int subtract(int x) {
        this.x -= x;
        return this.x;
    }
}
```

In this example, the value of the variable *x* is calculated by using the member variable, which means it can be easily modified. Thus, it is changeable and the function can be said to be carrying a side effect.

QUICK CHALLENGE

Create a chart to show the difference between pure and non-pure functions.

18.3.2 Higher-Order Function

In Java, a higher-order function can be replicated by adding a single or multiple lambda expressions in the parameters of a function. Additionally, the function must return a lambda expression. Let us see the following example.

```
public class HigherOrderFn {
    public <T> IFactory<T> createFactory(IProducer<T> prod, IConfigurator<T> conf) {
        // the lambda expression begins
        return () -> { // the arrow represents the lambda expression
            T ins = prod.produce();
            conf.configure(ins);
            return ins;
        } // the lambda expression ends
    }
}
```

Observe that a lambda expression is returned by method of `createFactory()`. For this reason, the *t* condition of a higher-order function is verified.

18.3.3 No State

There are no states in a functional program. Here, state means that the member cannot point (reference) to a variable of the object or class. Thus, it can only reference its own variables. For instance, a function with an external state is given as follows:

```
public class Subtract {
    private int beginningValue = 2;
    public int sub(int x) {
        // the original value is modified by addition
        return beginningValue + x;
    }
}
```

Now, observe another example where a function does not have an external state. Here, the values of the function cannot be changed.

```
public class Subtract {
    public int sub(int x, int y) {
        return x + y;
    }
}
```

18.3.4 Functional Interfaces

In Java, an interface with a single abstract method is called functional interface. As you know, abstract methods are those that are incomplete and are not implemented. Interfaces can have both static and default methods (implemented). A functional interface must have a single method which is not implemented. For example:

```
public interface TestingInterface {
    public void run();
}
```

Let's observe the following program:

```
public interface TestingInterface2 {
    public void run();
    public default void hello() {
        System.out.println("hello world");
    }
    public static void helloStatic() {
        System.out.println("static hello world");
    }
}
```

You must be thinking that since two methods that have been implemented, the interface may not be functional. But since the run method is abstract (i.e., it has not been implemented), our interface is still functional. Interfaces that have multiple abstract methods cannot be categorized as a functional interface.

18.4 | Object-Oriented versus Functional Programming



So far you have learnt object-oriented and functional programming. Now let us do the head-to-head comparison between them. Table 18.1 compares these two in multiple areas.

Table 18.1 Object-oriented and functional programming comparison

Parameter	Functional Programming	Object-Oriented Programming
Focus	It focuses on evaluation of functions	It focuses on objects
Data	It mainly uses immutable data	It mainly uses mutable data
Model	It follows declarative programming model	It follows imperative programming model
Parallel Programming	It supports parallel programming	It does not support parallel programming
Execution	Statements executed in any order	Statements executed in a specific order
Iteration	It uses recursion	It uses loops
Elements	Variables and functions	Objects and methods
Use	Used when there are a few things with large number of operations	Used when there are large number of things with a few operations
State	It does not care about any state	It does care about the state

(Continued)

Table 18.1 (Continued)

Parameter	Functional Programming	Object-Oriented Programming
Primary Unit	Function	Object
Abstraction Support	It supports abstraction over data and behavior	It supports abstraction over data only
Performance on Big Data Processing	Very good	Not so good
Conditional Statements	No support	Supports statements like if-else, switch, etc.
Example	<pre>employees.stream().filter(emp -> emp.salary < minSalary).forEach(emp -> System.out. println(emp));</pre>	<pre>for (Employee emp: employees) { if (emp.getSalary() < minSalary) } System.out.println(emp); }</pre>



How is Stream different from Collection?

18.5 | Lambdas

Now that you have understood the concept of functional interfaces, let us proceed to lambdas. When Java 8 was released, it gained significant traction with the introduction of lambdas. So what exactly are lambdas expressions?

Lambda expressions are an attempt by Java to enhance functional programming. A lambda expression assists a single or multiple instances of a functional interface via concrete implementations. Since lambdas are capable for concrete implementation of a single function, they are appropriate for functional interfaces. They can be created without necessarily adding them to a class. Lambda expressions can be treated as an object and can be passed and executed like an object. The format of a lambda expression is as follows:

parameter -> the body of expression

Here, the arrow operation is used to represent it.



Is using lambda faster than traditional Java programming?

18.5.1 Elements of Lambda Expression

Following are the main elements you need to know in order to code in Lambda.

18.5.1.1 Parameters

In a typical lambda expression which consists of multiple parts and each part is connected via an arrow. We will learn more about the meaning of these parts in the subsequent sections. On the left side, we can have n number of parameters, where n can also be a 0. Adding the type of parameter is optional. The type of parameter can be determined by the compiler by sifting through the coding context. Multiple parameters are entailed in round brackets, (). For single parameters, the use of a round brackets is optional. If there is no parameter, then it can be represented by an empty round bracket.

18.5.1.2 Body

Similar to the parameters, the body may also carry n number of statements. These statements are represented by curly brackets, {}. However, a single statement does not require curly brackets. The body expression also embodies the return type of the function.

Before going into lambdas, let us check the following example:

```
package java11.fundamentals.chapter18;
public class SimpleMethodExample {

    public void printDemo(String simpleText) {
        // A method to print the String
        System.out.println(simpleText);
    }
    public static void main(String[] args) {
        // Creating an instance of the object for our LamdaExample class
        SimpleMethodExample smEx = new SimpleMethodExample();

        // Assign a value to the object's string
        String simpleText = "A Simple String";
        smEx.printDemo(simpleText);
    }
}
```

The above program produces the following result.

A Simple String

You may have found the above example as a traditional approach where the implementation of the method is kept hidden from the caller. Here, the caller takes a variable which is then passed to a method `printDemo`. As you can see, there is a side effect issue.

Similarly, let us visit another example in which we are passing a behavior instead of a variable. We have utilized a functional interface for this example.

```
package java11.fundamentals.chapter18;
public class LambdaFunctionalInterfaceExample {

    interface printingSomeText {
        void print(String anyValue);
    }
    public void printLambdaText(String lambdaText, printingSomeText pst) {
        pst.print(lambdaText);
    }
    public static void main(String[] args) {
        LambdaExample lmd1 = new LambdaExample();
        String lambdaText = "Understanding Lambdas";
        printingSomeText pst = (String letsPrint)->{System.out.println(letsPrint);};
        lmd1.printLambdaText(lambdaText, pst);
    }
}
```

The above code produces the following result.

Understanding Lambdas

As you can observe, all the complexity behind the execution of printing the string has been handled by the interface. The method `printLambdaText` was used in the example only for the execution of the interface's block of code. Let us code the above program further:

```

package java11.fundamentals.chapter18;
public class LambdaFunctionalInterfaceExample2 {
    interface printingSomeText {
        void print(String anyValue);
    }

    public void printLambdaText(String lambdaText, printingSomeText pst) {
        pst.print(lambdaText);
    }
    public static void main(String[] args) {
        LambdaFunctionalInterfaceExample2 lmd1 = new LambdaFunctionalInterfaceExample2();
        String lambdaText = "Understanding Lambdas";

        printingSomeText pst = new printingSomeText() {
            @Override
            public void print(String anyValue) {
                System.out.println(anyValue);
            }
        };

        lmd1.printLambdaText(lambdaText, pst);
    }
}

```

The above program produces the following result.

Understanding Lambdas

Here, we wrote an implementation for our interface, which was then passed to the `printLambdaText` method. This example was necessary because it can help us realize why we need lambdas. Now, by introducing lambdas, we can recode by writing the following:

```

package java11.fundamentals.chapter18;
public class LambdaFunctionalInterfaceExample3 {

    interface printingSomeText {
        void print(String anyValue);
    }

    public void printLambdaText(String lambdaText, printingSomeText pst) {
        pst.print(lambdaText);
    }

    public static void main(String[] args) {
        LambdaFunctionalInterfaceExample3 lmd1 = new
        LambdaFunctionalInterfaceExample3();
        String lambdaText = "Understanding Lambdas";
        printingSomeText pst = (String letsPrint) -> {
            System.out.println(letsPrint);
        };
        lmd1.printLambdaText(lambdaText, pst);
    }
}

```

The above program produces the following result.

Understanding Lambdas

Does this not look better? By adding just a single line of a lambda expression, we have improved our code. Lambdas took the parameter and handled its mapping according to our method's parameter list. The code after the arrow can be seen as a concrete implementation of this method.

18.5.2 Examples

Let us code more examples to gain a better understanding of lambda expressions. We will implement the next example with the help of lambda expression, where we have an interface that draws a square.

```
package java11.fundamentals.chapter18;
interface drawSquare {
    public void drawIt();
}
public class LambdaDrawSquareExample {
    public static void main(String[] args) {

        int area = 5;

        // Lambda expression starts. An instance ds of the interface is taken as a parameter.
        Note that the absence of the parameters is represented by an empty round bracket
        drawSquare ds = () -> {
            System.out.println("The square is drawn for the area " + area);
        };

        // The concrete implementation provided by the lambda expression draws the square
        from the functional interface's method
        ds.drawIt();

    }
}
```

The above program produces the following result.

The square is drawn for the area 5

18.5.2.1 Lambda Expression without Parameters

We have already mentioned that a lambda parameter can be 0. We demonstrate this with the help of the following example:

```
package java11.fundamentals.chapter18;
public class LambdaExpressionWithoutParametersExample {
    interface announcement {
        public String announce();
    }

    public static void main(String[] args) {

        // Lambda expression begins
        announcement a = () -> {
            return "The flight going to New York has been cancelled due to the extreme weather";
        };
        // Lambda Expression ends

        System.out.println(a.announce());
    }
}
```

The above program produces the following result.

The flight going to New York has been cancelled due to the extreme weather

18.5.2.2 Adding a Single Parameter to the Lambda Expression

Now, let us see how we can use a parameter by continuing with our announcement example.

```
package java11.fundamentals.chapter18;

public class LambdaExpressionWithSingleParameterExample {

    interface announcement {
        // Adding a parameter to the method with a String variable
        public String announce(String annText);
    }

    public static void main(String[] args) {
        // First lambda expression begins
        announcement a1 = (annText) -> { // adding a single parameter with optional round
brackets
            return "We have an important announcement to be made. " + annText;
        }; // First lambda expression ends
        System.out.println(a1.announce("The flight going to New York has been cancelled
due to the extreme weather "));
        // Second lambda expression begins
        announcement a2 = annText -> { // adding a single parameter without using round
brackets
            return "We have an important announcement to be made. " + annText;
        };
        // Second lambda expression ends
        System.out.println(a2.announce("The flight going to Boston has been cancelled due
to a hailstorm "));
    }
}
```

The above program produces the following result.

We have an important announcement to be made. The flight going to New York has been cancelled due to the extreme weather
We have an important announcement to be made. The flight going to Boston has been cancelled due to a hailstorm

18.5.2.3 Lambda Expression with Multiple Parameters

Let us consider that we have to add the total marks of 5 subjects for 5 students. Now, we have an interface for the addition of marks where the method contains the value for subjects. Here, lambda expression can prove beneficial by providing convenience for concrete implementations. Since we have more than a single parameter, we must use round brackets in our expressions. Unlike single parameters, we cannot optionally remove brackets.

```

package java11.fundamentals.chapter18;

public class LambdaExpressionWithMultipleParametersExample {

    interface reportCard {
        // Adding multiple parameters to the method
        public int marksForSubjects(int mathematics,int physics,int biology,int history,
int chemistry);
    }

    public static void main(String[] args) {
        reportCard am1=(int mathematics,int physics,int biology,int history, int
chemistry)->(mathematics + physics + biology + history+ chemistry); // round brackets
are used for multiple parameters
        System.out.println(am1.marksForSubjects(74,87,66,53,90));

        reportCard am2=(int mathematics,int physics,int biology,int history, int
chemistry)->(mathematics + physics + biology + history+ chemistry);
        System.out.println(am2.marksForSubjects(40,40,50,60,70));

        reportCard am3=(int mathematics,int physics,int biology,int history, int
chemistry)->(mathematics + physics + biology + history+ chemistry);
        System.out.println(am3.marksForSubjects(50,60,70,60,70));

        reportCard am4=(int mathematics,int physics,int biology,int history, int
chemistry)->(mathematics + physics + biology + history+ chemistry);
        System.out.println(am4.marksForSubjects(64,68,71,67,70));

        reportCard am5=(int mathematics,int physics,int biology,int history, int
chemistry)->(mathematics + physics + biology + history+ chemistry);
        System.out.println(am5.marksForSubjects(85,86,55,75,88));
    }
}

```

The above program produces the following result.

370
260
310
340
389

18.5.3 Using return

Now that we have learned how to use various parameters for our use cases, let us move on to the `return` keyword. When a lambda expression contains a single statement, then it is optional to explicitly add or ignore the addition of `return` keyword. However, if there are multiple expressions in the lambda expression, then it is important to add the `return` keyword or you may face an error. Now, going by the above example of adding the marks of students, we can rewrite the program by adding the `return` keyword.

```

package java11.fundamentals.chapter18;
public class LambdaExpressionUsingReturnExample {
    interface reportCard {
        // Adding multiple parameters to the method
        public int marksForSubjects(int mathematics,int physics,int biology,int history,
int chemistry);
    }
    public static void main(String[] args) {
        reportCard am1=(int mathematics,int physics,int biology,int history, int
chemistry)->{
            return (mathematics + physics + biology + history+ chemistry); // adding the
optional return keyword
        };
        System.out.println("The total of the first student is "+am1.
marksForSubjects(74,87,66,53,90));
        reportCard am2=(int mathematics,int physics,int biology,int history, int
chemistry)->{
            return (mathematics + physics + biology + history+ chemistry);
        };
        System.out.println("The total of the second student is "+am2.
marksForSubjects(40,40,50,60,70));
        reportCard am3=(int mathematics,int physics,int biology,int history, int
chemistry)->{
            return (mathematics + physics + biology + history+ chemistry);
        };
        System.out.println("The total of the third student is "+am3.
marksForSubjects(50,60,70,60,70));
        reportCard am4=(int mathematics,int physics,int biology,int history, int chemistry)->{
            return (mathematics + physics + biology + history+ chemistry);
        };
        System.out.println("The total of the fourth student is "+am4.
marksForSubjects(65,56,95,65,78));
        reportCard am5=(int mathematics,int physics,int biology,int history, int
chemistry)->{
            return (mathematics + physics + biology + history+ chemistry);
        };
        System.out.println("The total of the fifth student is "+am5.
marksForSubjects(85,86,55,75,88));
    }
}

```

The above program produces the following result.

**The total of the first student is 370
The total of the second student is 260
The total of the third student is 310
The total of the fourth student is 359
The total of the fifth student is 389**

Now, let us modify this example by adding multiple statements in each lambda expression. We are adding a statement that can increase the marks of the subject of mathematics by 10 for each student. Without the `return` keyword, these expressions cannot be run.

```

package java11.fundamentals.chapter18;
public class LambdaExpressionWithMultipleParametersExample {
    interface reportCard {
        // Adding multiple parameters to the method
        public int marksForSubjects(int mathematics, int physics, int biology, int
history, int chemistry);
    }
    public static void main(String[] args) {
        // Multiple parameters in lambda expression
        reportCard am1 = (int mathematics, int physics, int biology, int history, int
chemistry) -> {
            mathematics += 10;
            return (mathematics + physics + biology + history + chemistry); // mandatory
inclusion of return
        };
        System.out.println("The total of the first student is " + am1.marksForSubjects(74,
87, 66, 53, 90));
        reportCard am2 = (int mathematics, int physics, int biology, int history, int
chemistry) -> {
            mathematics += 10;
            return (mathematics + physics + biology + history + chemistry);
        };
        System.out.println("The total of the second student is " +
am2.marksForSubjects(40, 40, 50, 60, 70));
        reportCard am3 = (int mathematics, int physics, int biology, int history, int
chemistry) -> {
            mathematics += 10;
            return (mathematics + physics + biology + history + chemistry);
        };
        System.out.println("The total of the third student is " + am3.marksForSubjects(50,
60, 70, 60, 70));
        reportCard am4 = (int mathematics, int physics, int biology, int history, int
chemistry) -> {
            mathematics += 10;
            return (mathematics + physics + biology + history + chemistry);
        };
        System.out.println("The total of the fourth student is " +
am4.marksForSubjects(65, 56, 95, 65, 78));
        reportCard am5 = (int mathematics, int physics, int biology, int history, int
chemistry) -> {
            mathematics += 10;
            return (mathematics + physics + biology + history + chemistry);
        };
        System.out.println("The total of the fifth student is " + am5.marksForSubjects(85,
86, 55, 75, 88));
    }
}

```

The above program produces the following result.

The total of the first student is 380
The total of the second student is 270
The total of the third student is 320
The total of the fourth student is 369
The total of the fifth student is 399

18.5.4 Lambdas with Loops

Lambdas are flexible and can also be integrated with loops. Let us consider two students, where each student is assigned the marks of 5 subjects. By using lambdas, we can use the **foreach** loop to add a print statement with each addition.

```
package java11.fundamentals.chapter18;
import java.util.*;
public class LambdaLoopExample {
    public static void main(String[] args) {
        List<Integer> student1 = new ArrayList<Integer>();
        student1.add(50);
        student1.add(60);
        student1.add(70);
        student1.add(80);
        student1.add(90);
        System.out.println("The marks of each subject of student1 :");
        student1.forEach((x) -> System.out.println(x));

        List<Integer> student2 = new ArrayList<Integer>();
        student2.add(90);
        student2.add(80);
        student2.add(70);
        student2.add(60);
        student2.add(50);
        System.out.println("The marks of each subject of student2 :");
        student2.forEach((x) -> System.out.println(x));
    }
}
```

The above program produces the following result.

```
The marks of each subject of student1 :
50
60
70
80
90
The marks of each subject of student2 :
90
80
70
60
50
```

18.5.5 Lambdas with Threads

We can also associate lambdas expressions with threads. Consider you have four threads that have different behaviors. Now, we will code the first thread without lambda expression. Observe how the second thread does the same thing with a better strategy using lambdas.

```

package java11.fundamentals.chapter18;
public class LambdaThreadExample {
    public static void main(String[] args) {
        Runnable run1 = new Runnable() {
            public void run() {
                System.out.println("The first thread is currently in the state of running");
            }
        };
        Thread t1 = new Thread(run1);
        t1.start();

        Runnable run2 = () -> {
            System.out.println("The second thread is currently in the state of running");
        };
        Thread t2 = new Thread(run2);
        t2.start();
        Runnable run3 = () -> {
            System.out.print("The id of the third thread is ");
            System.out.println(Thread.currentThread().getId());
        };
        Thread t3 = new Thread(run3);
        t3.start();

        Runnable run4 = () -> {
            System.out.print("The class of the fourth thread is ");
            System.out.println(Thread.currentThread().getClass());
        };
        Thread t4 = new Thread(run4);
        t4.start();
    }
}

```

The above program produces the following result.

**The first thread is currently in the state of running
 The second thread is currently in the state of running
 The id of the third thread is 14
 The class of the fourth thread is class java.lang.Thread**

18.6 | Date and Time API

The recent Java releases have improved the Date and Time API for tackling earlier issues that mainly existed in `java.util.Calendar` and `java.util.Date`.

18.6.1 Problems with Earlier APIs

In the past, developers complaint about the following conundrums:

1. Older versions of Calendar and Date APIs were found to be non-safe for threads. What this meant is that Java programmers had to continuously keep adding lines of code. Similarly, there were also issues pertaining to debug concurrency. Thus, it was problematic to debug several segments of the API simultaneously. The newer APIs in Java 9 for Date and Time are a game changer because they have been modified to be safe for threads while they are also plugged with immutability.
2. Older versions of Calendar and Date APIs lacked in design. Their built-in methods were ineffective in running standard tasks for applications. The Date and Time API of Java 9 is said to be ISO centric. All the values pertaining to time,

duration, periods, and date are handled through the use of consistent models. Likewise, standard tasks for daily applications can be performed easily via newer methods.

3. Earlier, time and resources went into the coding of logic related to the time zones of different areas. Newer APIs have automated this task through Local and ZonedDateTime APIs.

18.6.2 Local Date

The API of LocalDate adheres to the following ISO format:

```
yyyy-MM-dd
```

In the real world, such APIs are used in payroll systems for assigning a payroll to an employee at the beginning or end of the month. Likewise, in applications for storing birthdays such as in social media platforms, there are also requirements for dates. In case you require the latest date, type the following:

```
import java.time.LocalDate;
public class DateExample{
    public static void main(String args[]) {
        LocalDate currentDate = LocalDate.now();
        System.out.println(currentDate);
    }
}
```

The following output will be displayed.

2018-10-12

In order to get the result for any year, month, and day, you can use `of()` method. Similarly, `parse()` method can also be used. Let us see what happens if we use `of()` method:

```
import java.time.LocalDate;
public class DateExample{
    public static void main(String args[]) {
        System.out.println(LocalDate.of(2018, 01, 01));
    }
}
```

By using `parse()` method we can get the same answer:

```
import java.time.LocalDate;
public class DateExample{
    public static void main(String args[]) {
        System.out.println(LocalDate.parse("2018-01-01"));
    }
}
```

In both instances, we get the following answer.

2018-01-01

Likewise, let us go through a few more methods related to dates. Suppose you are developing an application which assigns tasks to employees. Now, an employee can be given a task that has to be done by the following day. In order to add this functionality in the code, you can write the following:

```

import java.time.LocalDate;
public class DateExample{
    public static void main(String args[]) {
        LocalDate taskDate = LocalDate.now().plusDays(1);
        System.out.println(taskDate);
    }
}

```

As such, you can change the values in the method to get your desired date in future.

Now, suppose you wish to go exactly one month back for your application requirements. This can be done through the `minus()` method.

```

import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
public class DateExample {
    public static void main(String args[]) {
        LocalDate lastMonth = LocalDate.now().minus(1, ChronoUnit.MONTHS);
        System.out.println(lastMonth);
    }
}

```

Consider the following example, where you have to check the date of a specific day. You can parse your date and utilize the method `.getDayOfWeek()` for displaying a date.

```

import java.time.DayOfWeek;
import java.time.LocalDate;
public class DateExample {
    public static void main(String args[]) {
        DayOfWeek whichDay = LocalDate.parse("2018-03-10").getDayOfWeek();
        System.out.println(whichDay);
    }
}

```

Similarly, the day of a month can be returned by using `.getDayOfMonth()` method, as follows:

```

import java.time.DayOfWeek;
import java.time.LocalDate;
public class DateExample {
    public static void main(String args[]) {
        int dayOftheMonth = LocalDate.parse("2018-03-10").getDayOfMonth();
        System.out.println(dayOftheMonth);
    }
}

```

In time related APIs, sometimes circumstances demand the need of identifying a leap year. Leap years can be checked by writing the following code:

```
import java.time.DayOfWeek;
import java.time.LocalDate;
public class DateExample {
    public static void main(String args[]) {
        boolean isItaLeapYear = LocalDate.now().isLeapYear();
        System.out.println(isItaLeapYear);
    }
}
```

For comparing two dates to check when a date occurred in comparison to another date, we can write:

```
import java.time.DayOfWeek;
import java.time.LocalDate;
public class DateExample {
    public static void main(String args[]) {
        boolean beforeOrNot = LocalDate.parse("2018-06-13")
            .isBefore(LocalDate.parse("2018-06-10"));
        System.out.println(beforeOrNot);
        boolean afterOrNot = LocalDate.parse("2018-06-10")
            .isAfter(LocalDate.parse("2018-06-09"));
        System.out.println(afterOrNot);
    }
}
```

18.6.3 LocalTime

The `LocalTime` class returns time. It works similar to the `LocalTime` class. For getting the current time, write the following:

```
import java.time.LocalTime;
public class TimeExample{
    public static void main(String args[]) {
        LocalTime whatIsTheTime = LocalTime.now();
        System.out.println(whatIsTheTime);
    }
}
```

To parse time, you can write:

```
import java.time.LocalTime;
public class TimeExample{
    public static void main(String args[]) {
        LocalTime parsingTime = LocalTime.parse("03:20");
        System.out.println(parsingTime);
    }
}
```

The same thing can be done by using `of()` method:

```
import java.time.LocalTime;
public class TimeExample{
    public static void main(String args[]) {
        LocalTime usingOf = LocalTime.of(3,20);
        System.out.println(usingOf);
    }
}
```

To add time, you can use the `.plus()` method. We have used this method to add 5 hours to the current time.

```
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;
public class TimeExample{
    public static void main(String args[]) {
        LocalTime fastForward = LocalTime.parse("03:20").plus(5, ChronoUnit.HOURS);
        System.out.println(fastForward);
    }
}
```

To get the hour, we can write:

```
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;
public class TimeExample{
    public static void main(String args[]) {
        int whichHour = LocalTime.parse("03:20").getHour();
        System.out.println(whichHour);
    }
}
```

To compare the time, we can write:

```
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;
public class TimeExample{
    public static void main(String args[]) {
        boolean comparingTime = LocalTime.parse("03:20").isBefore(LocalTime.
parse("02:30"));
        System.out.println(comparingTime);
    }
}
```

Sometimes in DB queries, records are required according to a given time period. To obtain such records, we can get time for noon, minimum, and maximum:

```
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;
public class TimeExample{
    public static void main(String args[]) {
        LocalTime maximumTime = LocalTime.MAX;
        System.out.println(maximumTime);
    }
}
```

18.6.4 LocalDateTime

While the above classes are useful for specific cases pertaining to date and time, sometimes both values are required (i.e., a date as well as the exact time for that day). For such scenarios, `LocalDateTime` is used. Now, let us go through its methods. To get the current date and time, we have to write:

```
import java.time.LocalDateTime;
import java.time.LocalTime;
public class TimeExample{
    public static void main(String args[]) {
        System.out.println(LocalDateTime.now());
    }
}
```

To use `of()` method, we write:

```
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
public class TimeExample{
    public static void main(String args[]) {
        System.out.println(LocalDateTime.of(2018, Month.MARCH, 10, 03, 30));
    }
}
```

The same thing can also be done for parsing:

```
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
public class TimeExample{
    public static void main(String args[]) {
        System.out.println(LocalDateTime.parse("2018-01-20T06:24:00"));
    }
}
```

For adding and subtracting time, we can use `plus()` and `minus()` methods, just as we have been using them previously.

```

import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
public class DateAndTimeExample {
    public static void main(String args[]) {
        LocalDateTime addHours = LocalDateTime.now().plusHours(3);
        System.out.println(addHours);
    }
}

```

Now, let us see the minus example.

```

import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
public class DateAndTimeExample {
    public static void main(String args[]) {
        LocalDateTime subHours = LocalDateTime.now().minusHours(3);
        System.out.println(subHours);
    }
}

```

Lastly, for getting a specific month, we can simply write:

```

import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
public class DateAndTimeExample {
    public static void main(String args[]) {
        System.out.println(LocalDateTime.now().getMonth());
    }
}

```

18.6.5 ZonedDateTime API

In order to combat the issue of time zone, we can use `ZonedDateTime()` in Java 9. In this API, an identifier called `ZoneID` represents time zones. To create the time zone of any specific city, type the following code in the IDE:

```

import java.time.ZoneId;
public class TimeZoneExample {
    public static void main(String args[]) {
        ZoneId id = ZoneId.of("Asia/Seoul");
        System.out.println(id);
    }
}

```

To get all the time zones that are listed in the API, we can simply write:

```
import java.time.ZoneId;
import java.util.Set;
public class TimeZoneExample {
    public static void main(String args[]) {
        Set<String> allIds = ZoneId.getAvailableZoneIds();
        System.out.println(allIds);
    }
}
```

To choose a specific time zone, we can write:

```
ZonedDateTime specificZone = ZonedDateTime.of(localDateTime, zoneId)
```

Now, let's create a localDateTime:

```
import java.time.LocalDateTime;
import java.time.Month;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.Set;
public class TimeZoneExample {
    public static void main(String args[]) {
        LocalDateTime ltd = LocalDateTime.of(2018, Month.MARCH, 10, 07, 20);
        System.out.println(ltd);
    }
}
```

Now, it is possible to add four hours and create a ZoneOffset in the above example. Let us continue the code.

```
import java.time.LocalDateTime;
import java.time.Month;
import java.time.OffsetDateTime;
import java.time.ZoneId;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.util.Set;
public class TimeZoneExample {
    public static void main(String args[]) {

        LocalDateTime ltd = LocalDateTime.of(2018, Month.MARCH, 10, 07, 20);
        ZoneOffset os = ZoneOffset.of("+04:00");
        OffsetDateTime osbyfour = OffsetDateTime.of(ltd, os);
        System.out.println(osbyfour);
    }
}
```

The result of localDateTime of method is as follows

2018-03-10T07:20+04:00



Can DateTimeAPI get a user's local date?

Summary

Functional programming is a practice of programming as functions like mathematical functions. Lambda expression is as a way of supporting functional programming in Java. This language is a declarative one, which means logic is expressed without its control flow.

In this chapter, we have learned the following concepts:

1. Functional programming and lambda.
2. Higher order and first order functions.
3. Pure functions and how to use them.
4. Recursion and how to use it in program.

In Chapter 19, we will learn about multithreading and reactive programming. We will learn about the multithreading world and understand the important concepts of concurrency. We will explore various examples and learn about deadlocks, synchronization blocks, lazy initialization, etc.

Multiple-Choice Questions

1. We need to compile Lambda expression to anonymous inner classes.
 - (a) True
 - (b) False
2. The filter method in Stream API takes in a _____ as argument.
 - (a) Predicate
 - (b) Consumer
 - (c) Function
 - (d) Supplier
3. Functional interfaces can have more than one default methods.
- (a) (b) (c) (d)
4. Which of the following is used to get only the current time?
 - (a) LocalDate.now();
 - (b) LocalTime.now();
 - (c) LocalDate.now()
 - (d) All of the above
5. Can lambda expression accept multiple parameters?
 - (a) Yes
 - (b) No

Review Questions

1. What is lambda?
2. How does functional programming work?
3. What are the benefits of using lambdas?
4. How do we print date in a local format?
5. How do we print date according to time zone?
6. How do we use lambdas with threads?
7. How do we use lambdas with loops?

Exercises

1. Write a program to print stock prices of top 10 companies using lambda and functional programming.
2. Write a program to use DateTime API to print minute-by-minute status of a football match.
3. Create a chart to present all the benefits of using lambdas and functional programming.

Project Idea

Create a chat application using lambda and functional programming. Capture each conversation and use DateTime API to print the conversation as per the user's local time. Consider user's time zone in this case.

Recommended Readings

1. Kishori Sharan. 2018. *Java Language Features: With Modules, Streams, Threads, I/O, and Lambda Expressions*. Apress: New York
2. Pierre-Yves Saumont. 2017. *Functional Programming in Java: How Functional Techniques Improve Your Java Programs*. Dreamtech Press: New Delhi
3. Raoul-Gabriel Urma, Mario Fusco, Alan Mycroft. 2018. *Modern Java in Action: Lambdas, streams, functional and reactive programming*. Manning Publications: New York

Multithreading and Reactive Programming

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Reactive programming.
- Multithreading and how to program.
- Concurrency API improvements.
- How to deal with individual threads.
- Synchronization blocks.
- Deadlocks and how to resolve deadlocks.
- Concurrent data structures.
- How to design concurrent Java programs.
- Controlling sequential executions.
- How to avoid lazy initialization.

19.1 | Introduction

Java 11 is the latest version of Java released by Oracle, which has some interesting features. Before its release, Java 9 brought major updates. Java 9 updates offer improved support to large heaps as there is great demand for improving the capacity of Java to handle large memory sizes that are required to support cloud applications. It follows a module system and includes several Java projects that were initially defined for the new release.

Our focus in this chapter remains on discussing the improved multithreading capabilities of Java. We will begin this chapter by describing concepts such as reactive programming and multithreading, especially with their application using the new and improved Java's concurrency utilities. We will also cover thread transition, thread interaction, and `newworkstealingPool()` method. With the exercises and other tools provided in this chapter, we believe that Java programmers will learn to use multiple threads in their programming and ensure that their programs are efficient by employing the available capacities in modern processors.

19.2 | Reactive Programming



Reactive programming is a special method that employs an asynchronous style. It refers to a method that employs improved control over data streams and data streams use in changing the way the programming behaves with the future data stream. All kinds of data streams can be expressed using reactive methods and the programs can be designed to execute different changes automatically, according to the data flow that they receive from program outputs and other important parameters.

Reactive programming works well when employed in an imperative setting. The use of relations and the effective change in program behavior is high possibility in Java. In fact, any language that can directly control and describe hardware components such as Verilog can benefit from the use of reactive programming. It gives improved control over the available hardware resources.

Reactive programming works well by understanding that there is a publisher that keeps producing data, while there is also a subscriber that requires asynchronous access to the process information. This relationship is best described by Figure 19.1.



Figure 19.1 Publisher and subscriber relationship for reactive programming.

There are several benefits to implementing reactive programming. Here are two common advantages that you get as a reactive programming expert:

1. You can use a simpler code for the required tasks. This allows you to create a readable code, which is easy to implement as well as improve when required in a client application.
2. It allows programmers to focus on implementing business logic in their programming solutions. This takes them away from following a conventional boilerplate code to stay away from similarity and come up with unique programming solutions.

There are four key attributes of a reactive system, which are described in the reactive manifesto. This is illustrated in Figure 19.2.

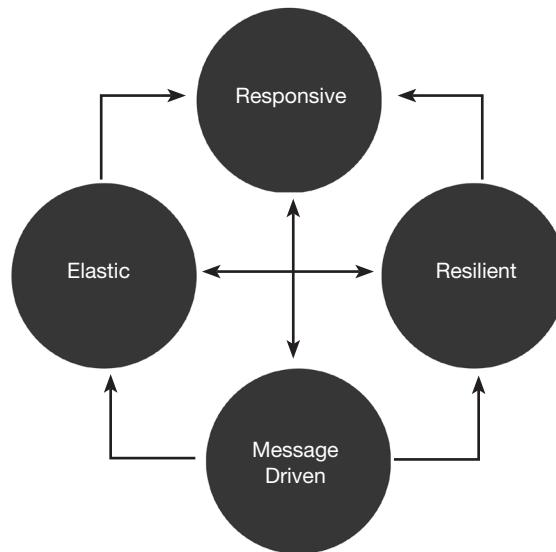


Figure 19.2 Four key attributes of reactive systems.

1. **Responsive:** The system should respond in a timely manner, it not only ensures usability but also effectiveness.
2. **Resilient:** The system should respond in case of failure as well. This can be achieved by focusing on containment, replication, delegation, and isolation.
3. **Elastic:** The system should stay in elastic state. In other words, it should respond to varied workload. In the case of high workload, resource allocation should increase. In case of low workload, resources can be released.
4. **Message driven:** The system should ensure loose coupling, isolation, and location transparency by establishing boundaries between components and relying on asynchronous message passing.



What is the use of “reactivity” in Java?

Concurrency issues are better controlled with reactive programming techniques. The need for creating low-level threads that often depend on ideal synchronization is significantly reduced, thereby resulting in the creation of an environment which is conducive to efficient programming.

Reactive programming improves the efficiency associated with memory use. It is possible to create different streams that are then implemented with a multithreading mechanism to further enhance the results of employing reactive programming through the ideal APIs.

Reactive programming is a successful model with the ability to offer great results in all kinds of programming problems. It is not limited to only offer improvements in real-time applications, as it can be implemented to create reactive hardware definitions that use multithreading schemes.

This model is great for introducing powerful functions that use the available processors to carry out data transformations. There is no need to change either the subscriber or the publisher in the programming model. There can be an N number of processors that can work on incoming data streams and then provide results to the subscriber according to the particular needs of the application. This technique ensures that programming nodes can be made independent and have the capacity to handle any situation by simply changing the rules that work on the received data streams issued by the publisher.

Java 11 is specially designed to facilitate cloud applications that run in a real-time environment. This means that the language must provide resources that allow for a speedy process and ensure that it is possible to provide the best output, according to the dynamic inputs controlling the situation.

Reactive programs are interactive and are ideal to deal with the changing environment, which is the need in a data center providing support to cloud applications. However, reactive programming through Java interface also has several other applications. It certainly has the ability to create hardware drivers, provide a better virtual machine, and improve protocols that handle data streams.



How will you lose stream data in case of accessing it via concurrent program?

19.3 | Reactive Programming

Since Java 9 update, Java has great potential to carry out reactive programming. In fact, the reactive style is the best one that you can use in Java as it works well with conditions where you can declare the direction that the program must take in the case of receiving specific inputs and processing requests. Since Java 9 update, Java has Flow API that allows the use of reactive streams. This allows programmers to create and filter observable objects that can then be used to implement a dynamic behavior change, whenever it is required in any setting.

Spring is a popular Java framework in this regard, which can be employed for implementing strong reactive patterns and creating applications with the ability to show resilient behavior and improved performance. Throughout the course of this chapter, we will describe various resources and show relevant examples that will allow you to use several APIs to implement reactive programming that performs well in dynamic needs. These are the needs that are specifically required when working with cloud-based applications that need to use large heap sizes and perform better when there are increased processing needs.

The Flow APIs in JDK 9 and after are now working according to the Reactive Streams Specification. There are various implementations which support this standard, while the main goal remains the application of the reactive programming framework that was described earlier.

The API uses a model which depends on a push and a pull model. The Observer is a push mode, where source data items are pushed to reach the application. On the other hand, the pull comes from the Iterator in the Flow API, where the application actively pulls items present at the data source. The API runs by first requesting N data items and the publisher then pushes a maximum of these N items to the subscriber. The forwarded items may be less if required. The Flow API therefore performs by carrying out a mixture of pull and push steps for reactive functionality. Here is an example of how this API may function, which was taken from the Oracle Community Site (<https://community.oracle.com/docs/DOC-1006738>):

```

public static interface Flow.Publisher<T> {
    public void subscribe(Flow.Subscriber<? super T> subscriber);
}
public static interface Flow.Subscriber<T> {
    public void onSubscribe(Flow.Subscription subscription);
    public void onNext(T item) ;
    public void onError(Throwable throwable) ;
    public void onComplete() ;
}
public static interface Flow.Subscription {
    public void request(long n);
    public void cancel() ;
}
public static interface Flow.Processor<T,R> extends Flow.Subscriber<T>,
Flow.Publisher<R> {
}

```

This describes the flow process of a typical Java API that employs reactive programming. It uses the pull and push phases that were discussed. We will further present how to employ the individual functionalities from the subscriber and the publisher that are present at the operating end of a Java code that uses reactive programming streams and principles.

QUICK CHALLENGE

Take the example of a bank ATM and write a program using the reactive programming concept which allows user to withdraw cash from any ATM.

19.4 | What is Multithreading?



In the context of computer processing, multithreading refers to the capability of using multiple execution threads that can occur independently of each other through a unified process. The threads use the same pool of resources but have different bits of information that require processing (including exception handles), the CPU register requirements, and the stacking status in the addressing space.

The use of multiple threads is an excellent approach for empowering processes on a single processor system. This allows the use of at least two threads where one can always be responding to the user, while another one may be in working condition. However, since most modern computers have multiple processors, the power to employ multiple threads creates an ideal concurrency solution.

Multithreading is also associated with the need to program in a careful manner. It can cause deadlock and conditions where the processor finds it difficult to make the ideal execution decisions. 64-bit operating systems such as Windows often employ pre-emptive multithreading where the software is responsible for switching between different threads. This allows for switching to a high priority thread while using a low priority thread as the trigger element. Another method is the use of cooperative multithreading, which is extremely powerful but creates deadlocks if there is any problem during the execution of a single thread.

Most multithreading occurs at a blistering pace, in which the available time slices into many pieces in the fraction of a second and queued for different threads. This gives the impression to the user that all the threads are running in parallel, when in actual reality, they are taking turns during the processing cycles which is too fast for a typical application.

The primary advantage of using this method is to efficiently employ the available computational resources. A single thread may not often employ the available cache in a physical system. With the presence of multiple threads, it is possible to use the available CPU resources more efficiently, because most of them depend on the results of the current execution.

Running different threads ensures that the resources do not remain idle and will be employed by one or the other thread from time to time, within the microsecond execution cycles. However, multiple threads may interfere with each other if their use is not carefully coordinated. They also face problems if used at lower frequencies. Modern computers have significantly reduced these problems and provide an ideal scenario in which multithreading can be employed with improved accuracy and less failures.

Hardware manufacturers such as Intel claim that the use of multithreading can cause a 30% improvement in the performance of their processors and related components. Processes that often require floating point operations may gain as much as double the performance with parallel processing.

This means that with the right exposure allowed to software elements, multithreading can significantly reduce the execution time of a computer program, allowing for swift processing. It improves user experience and provides important processing advantages with reduced stop-times during executions.

19.4.1 Multithreading in Java

Java is often employed to create programs that may serve several users through a single supporting platform. The thread in Java can be best defined as the smallest processing unit which can be employed to implement multitasking in a program environment. All threads share common heap, thereby ensuring efficient use of the available memory. The switching occurs according to a well-defined context which always happens faster than the time required for the processing of a thread.

Multithreading in Java is an excellent way of improving the performance of programs that produce animations and complex decisions such as games. It is an excellent technique because it can produce the simultaneous results required in such programming environments. You get the following advantages when using multithreading in Java:

1. It is possible to run multiple operations in your program at once, thus reducing the overall time required to produce a response for the program user.
2. Threads are independent in terms of their program decision-making, which allows the user to always have an available interface.
3. Independent threads are excellent for handling program execution exceptions. Stopping one thread does not stop the entire execution and simply alters the resource allocation to produce excellent user interaction.

Understanding the life cycle of a Java thread is important. A thread goes through four separate phases (i.e., if we do not count the running phase), which actually does not affect the thread itself. The four phases are:

1. New phase.
2. Runnable phase.
3. Non-runnable phase.
4. Terminated.

Figure 19.3 is a visual presentation of how these phases interact with each other, including the actual running of the thread.

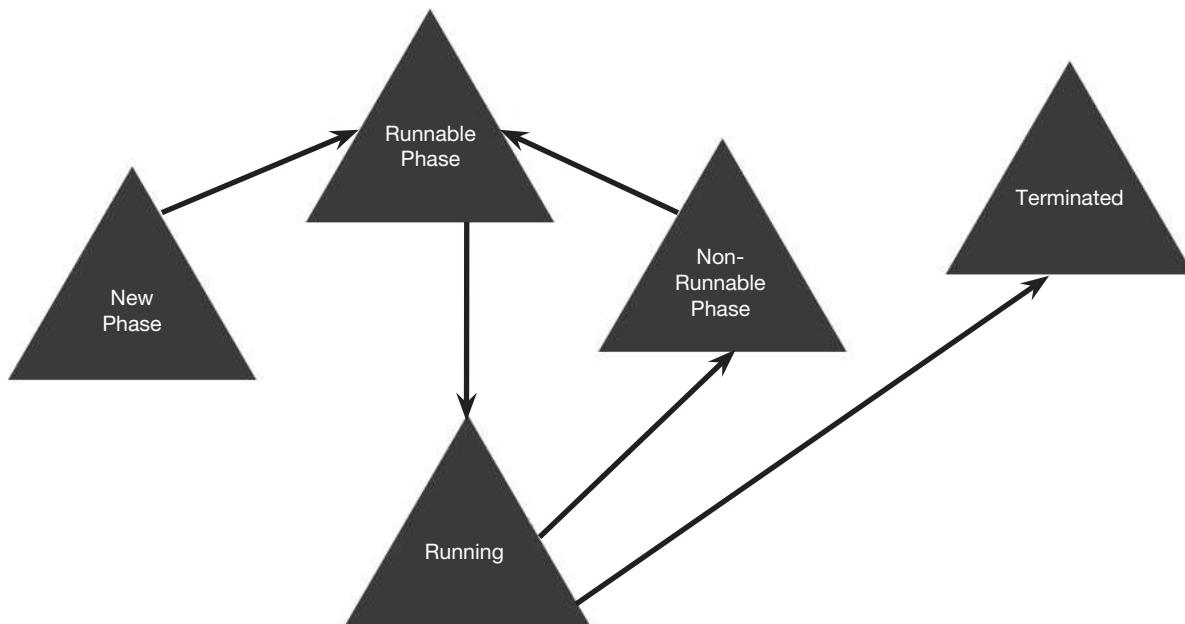


Figure 19.3 Thread life cycle.

The new phase is the initial phase of any thread, which is created when the Java program calls for a Thread class instance. However, this is the phase of the project before the invoking of the `start()` method that produces the next phase.

With the above method employed in the code, the thread is now ready for execution. This is termed as *runnable phase*. However, the actual running of the program is based on when the thread scheduler places the thread on a processing schedule.

The thread that enters the *non-runnable phase*, where it is blocked from further additions. This is important because the thread is still alive and any modification or further processing can cause program errors. The termination is identified when the `run()` method exists for the thread. This is also termed as *dead state*.

QUICK CHALLENGE

Explain the thread life cycle concept using a real-life example of a bank ATM in which the user can withdraw cash from any ATM.

See Figure 19.4 to understand the thread states from a different view.

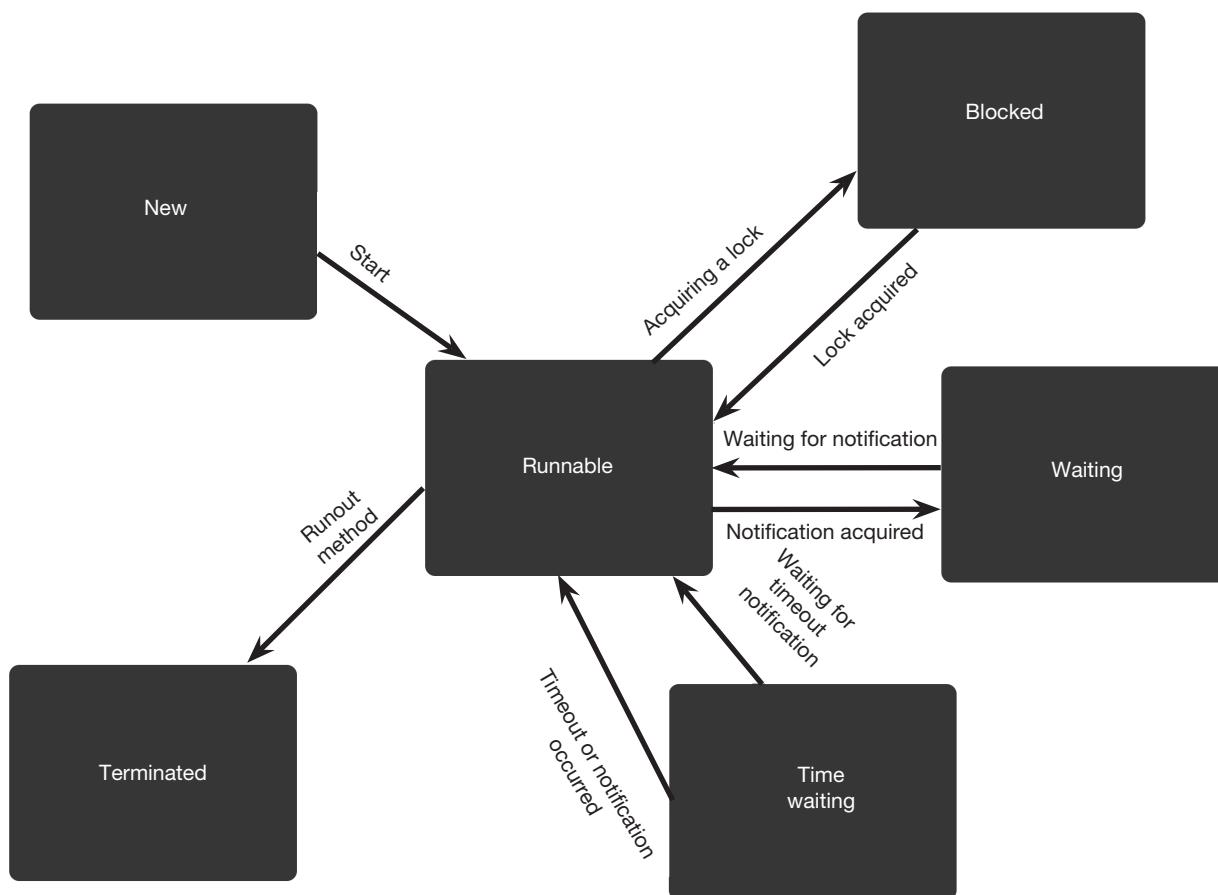


Figure 19.4 Transition of threads from different phases.

19.4.2 Programming with Multithreading

A thread is created in Java by either extending the Thread class or starting the Runnable interface. The use of the Thread class provides the required methods and constructors that contain the operations that must be performed on the objects of the class. There are several constructors such as `Thread()`, `Thread(Runnable r, String name)` that are employed for creating threads.

Methods such as `join()`, `start()`, `run()`, `sleep()`, `getPriority()`, and `setPriority(int priority)` are employed in the Thread class objects. Other methods may include testing the thread status, returning the thread id, or even

modifying the name of the thread. The `start()` method initiates a new threat and sets it up in the calling stack. Once the thread is selected for processing, it always executes its `run()` method to perform the required functionality. Here is an example of a functioning Java thread:

```
package java11.fundamentals.chapter19;
public class JavaMultiThreadingExample extends Thread {
    public void run() {
        System.out.println("My newThread is Running");
    }
    public static void main(String args[]) {
        JavaMultiThreadingExample newThread = new JavaMultiThreadingExample();
        newThread.start();
    }
}
```

The above program produces the following result.

My newThread is Running

This is a simple thread that will print the message, “My newThread is Running”. It creates a single class instance with one `run()` method. This method extends the `Thread` class to carry out the intended functionality. Another way to perform the same functionality is to create runnable instances as presented in the following example:

```
package java11.fundamentals.chapter19;
public class JavaMultiThreadingWithRunnableExample implements Runnable {
    public void run() {
        System.out.println("My newThread is Running");
    }
    public static void main(String args[]) {
        JavaMultiThreadingWithRunnableExample myRunnableObj = new
JavaMultiThreadingWithRunnableExample();
        Thread newThread = new Thread(myRunnableObj);
        newThread.start();
    }
}
```

The above program produces the following result.

My newThread is Running

This one implements the `Runnable` interface. Since you are not extending, you need to show the creation of an explicit `Thread` class object, which is `th1` in this example. Another important concept is the thread scheduler. It is an important Java virtual machine (JVM) component that decides which of the runnable threads will be chosen next for execution. It can employ the method of time slicing or pre-emptive scheduling. Time slicing treats all threads equally while the pre-emptive scheduling allows for the setting of priority in the available threads.

In this regard, the `sleep()` method holds important value in Java. It pauses the thread for the mentioned milliseconds, which may be defined. The Java thread scheduler will ignore the thread and move on to the next one if it finds a selected thread with a running `sleep()` method. Here is an example of its use:

```

package java11.fundamentals.chapter19;
public class JavaMultiThreadingWithSleepExample extends Thread {
    public void run() {
        for (int i = 1; i < 4; i++) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
            System.out.println(i);
        }
    }
    public static void main(String args[]) {
        JavaMultiThreadingWithSleepExample myThread1 = new
JavaMultiThreadingWithSleepExample();
        JavaMultiThreadingWithSleepExample myThread2 = new
JavaMultiThreadingWithSleepExample();
        myThread1.start();
        myThread2.start();
    }
}

```

The above program produces the following result.

1
1
2
2
3
3

The sleep method throws an exception whose status can then be found with catch and then printed. There are two threads, and the running of each thread then produces a sleep period of 500ms. This means that myThread1 will get ignored in the next cycle and myThread2 will get picked up for processing. This will result in this program printing two 1s, two 2s, and two 3s as the console output.

All threads in Java can only run successfully just one time. If a thread is wrongly called multiple times, all other instances after the first one will return an illegal thread exception error. If the `run()` method is directly employed without first initiating the start, it produces the addition of the thread on the same call stack. The direct running of Thread objects is of no use, as they lose their functionality as individual threads and are simply treated as normal code objects.

Synchronization is also possible with the use of `join()` method. This is a method that stops a thread from executing until another referenced thread has been truncated. This is excellent for creating synchronized code, which may be important in several Java programs. You can also find out information about the thread which is currently running by using the `currentThread()` method. Changing the name is also possible, but it only holds value when this is used with other programming elements to produce the required functionality in a specific program.

The next important topic in this regard is to understand how the thread scheduler sets up the priority of the available threads. Java provides a priority value to every thread in a program. This is presented as a number ranging from 1 to 10. Normally, Java scheduler uses pre-emptive scheduling where the threads are arranged for execution according to these priority values. Although this may not be the case if a particular JVM is performing time slicing of multiple threads.

There are three important constants that are present in the Thread class. They describe the minimum, normal, and maximum priority for threads in the following forms:

1. `public static int MIN_PRIORITY`
2. `public static int NORM_PRIORITY`
3. `public static int MAX_PRIORITY`

The normal priority with a value of 5 is always selected for a thread as a default value. The `MIN_PRIORITY` directly sets the priority to 1 (lowest), while `MAX_PRIORITY` sets it up to 10 (highest).

An interesting point to note is that the JVM often creates a Daemon thread just to help the user created threads by providing them support and functionality benefits. Such a thread is automatically terminated by the JVM when all user threads are processed.



How will you guarantee thread priority?

Java also employs pool threads that are fixed thread objects that can be used for specific support actions. This allows for a quicker method, as creating user-defined Thread objects takes more processing time. You can also create multiple threads with a single object.

This is possible with the `ThreadGroup` class in Java. Multiple threads can be implemented within a single object of this class. All groups and their individual threads can be named in a customized manner. Here is an example that describes how the `ThreadGroup` may be set up. In an actual use, the described functionality may be complex, with each thread requiring specific processing power.

```
package java11.fundamentals.chapter19;
public class JavaMultiThreadingThreadGroupExample implements Runnable {
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String[] args) {
        JavaMultiThreadingThreadGroupExample runnable = new
JavaMultiThreadingThreadGroupExample();
        ThreadGroup myThreadGroup = new ThreadGroup("My Thread Group");
        Thread t1 = new Thread(myThreadGroup, runnable, "My First Thread");
        t1.start();
        Thread t2 = new Thread(myThreadGroup, runnable, "My Second Thread");
        t2.start();
        Thread t3 = new Thread(myThreadGroup, runnable, "My Third Thread");
        t3.start();
        System.out.println("My Thread Group Name: " + myThreadGroup.getName());
        myThreadGroup.list();
    }
}
```

The above program produces the following result.

My First Thread
My Third Thread
My Thread Group Name: My Thread Group
My Second Thread
<code>java.lang.ThreadGroup[name=My Thread Group,maxpri=10]</code>
Thread[My Second Thread,5,My Thread Group]

This is a program that prints the name of the individual threads of the group as first, second, and third, and then the thread group name is returned as the Parent Thread Group after the first three threads are executed. The `list()` method then prints out the complete information of the created thread group `tg1`.



What will happen if two threads access the same resource at the same time?



19.5 | Concurrency

Concurrency is an important concept in computer programming. It defines the ability of a program to be executed in such a manner that any change in its running order does not affect how the final output is produced by it. It can include parallel processing of different processing units as well, which may be concurrent with each other and can be executed without affecting each other.

This improves the performance, especially in modern computers that have multiple processors with the ability to run multiple threads at the same time, which are independent of one another. The concurrency requires the decomposition of a program in partial elements so that these parts can be concurrently executed to use the available resources with improved efficiency.

Concurrency can certainly lower the program execution time and especially provide support in scenarios where an application may slow down due to a build-up of threads that may be run when performing sequential processing. All languages employ complex mathematics to create highly specialized and efficient concurrent processing schemes to improve program execution.

As the name suggests, the use of concurrency in computer programs results in various computations, which all overlap one another. The need for the sequence is eliminated, although a single program may use both sequential and concurrent program execution. Concurrency allows the application of modular programming, where the results from different computations can all be combined to produce the desired program functionality.

19.5.1 Advantages of Concurrency

The use of concurrency in Java programs can offer various benefits to programmers. Here are the top advantages that you can get with concurrent programming practices:

1. There is improvement in response of the programs. The program elements do not have to wait for specific input/output computation results, allowing the processor resources to be fully employed, while one thread is waiting for the results of a specific processing function.
2. There is improvement in the throughput of the program by a considerable margin. This is possible because parallel execution takes place constantly. This results in more tasks getting completed in every processing or execution cycle in the program. It can be simply described as a measure that shows improved resource efficiency.
3. There is improvement in the creation of better program structure. Concurrent programming is excellent for most problem domains that may require the result of several individual processes to finally reach conclusion. This may be the case where a program performs a complex function, which will often depend on the calculations of several individual elements that can be executed in any possible sequence.

Concurrent programming can use different methods. An ideal way to implement concurrent programming is multithreading. A set of threads allows an operating system or a JVM to use multiple processors to run parallel executions and therefore, gain the benefits of concurrency. Java is a language that uses explicit communication to describe the use of concurrent programming.

Java and C# are important programming languages that use communication during concurrent components that share memory resources. This is accomplished by setting up threads that can coordinate through locking mechanisms. A program that has the required functionality to avoid problems between parallel threads is termed as *thread-safe product*.

There are other methods, but we are not going to discuss them since they are not associated with the concurrency employed through specific APIs. Java implements the concept of concurrency with the use of the Thread class that we have explained above, as well as employing runnable instances which provide control over the execution of different threads.

19.5.2 Concurrency in Java

For typical computer users, they always expect that their computers are going to perform multiple tasks at the same time. They want their word processor to accept their keyboard inputs and display them on the screen, while still listening to songs that are directly streamed from the Internet. This functionality is only possible with the use of concurrent software.

Java has been supporting concurrency since its version 5.0, and the latest version is a lot more powerful. It does not provide the basic concurrency support in its JDK and class libraries, it also has an excellent collection of concurrency APIs that can perform at the highest level. The `java.util.concurrent` package contains powerful APIs that allows programmers to implement advanced concurrency and increase the throughput performance of their programs.

Java uses two units in concurrent programming: threads and processes. Most concurrent programming functions are delivered and performed on threads. Sometimes processes are also employed during concurrency assignments. A computer system often has a host of open processes and threads at any given time, regardless of the number of cores on the processor. Take the example of an operating system, where you can check the number of processes that are operational at the same time.

Understanding processes and threads in Java is ideal for understanding how concurrency works, especially in Java 9 update which has specific concurrency API improvements. A process is defined as a set of execution resources that have specific heap space. Processes are often defined as applications or programs, although it is possible that a single user application may be running multiple processes in the background.

The JVM mostly employs a single process, when using the computational resources of the hosting computer. However, Java provides support for creating multiple processes as well. It is possible with the use of a special `ProcessBuilder` object, which is a specific application beyond the scope of our concurrency article.

Threads, on the other hand, are small (lightweight) processes. They are simply termed as the most basic unit of execution that is delivered to the physical processor by JVM. Threads are also a part of the execution, but it is easier to prepare them as fewer resources are required. They always occur within a singular process in Java, while each process in any application would always have a single thread.

All threads share the resources that are allocated to the overall process in Java. This includes the heap which is assigned to the JVM as well as the open files, according to the libraries mentioned in the program bytecode during executions. Although this creates an efficient recipe, it is harder to control the use of resources without using an effective communication system for inter-thread messages.

The Java platform allows the use of multiple threads, as they are always present, even if your program only asks for one. The system also creates its own threads to perform important functions like built-in memory management. The programmer must only focus on the main thread, which is the one that contains program instructions.

Now, it is important to understand how the JVM can employ them in order to improve the performance of a Java program. Most Java applications can create multiple threads that may then be used for parallel processing or for implementing asynchronous behavior in the program.



Is there any alternative to concurrency? Explain.

Concurrency is the technique which ensures that all tasks can be speeded up by breaking them into smaller tasks that can occur independently of one another and executed in parallel in different threads. The performance of Java Runtime depends strongly on the results of the current parts that are getting processes. The Amdahl's Law governs the maximum performance gain that can be achieved with this practice as:

The F denotes the percentage of the program that must run in a synchronized manner and cannot go through parallel processing and N is the total number of processes that are running at any given time. Concurrency is not free from its own problems. This happens because threads can control their call stack, but share heap locations with each other. The first problem that concurrency produces is that of visibility while the second one is about the ideal access.

The first problem of visibility occurs when the data shared by the first thread is then used and changed by another thread without informing the first thread about it. This means that when the elements of the first thread go in their next processing, the change of data completely eliminates the functionality of the program and produces wrong data inputs within them.

The second problem of access occurs when multithreading and parallel processing is implemented during a concurrency run. This means that two different threads may attempt to access the shared data at the same time, while changing it after their particular processing in a single go. This creates a deadlock as the program cannot comply with multiple demands to access and change the same data location from two different threads at the same time.

The `java.util.concurrent` API package is important in this regard as it provides the support required for creating different threads and implementing strong measures for the required code synchronization. It is important that threads that are sharing data sources must run in an organized manner, where it is not possible to corrupt the data for the other thread.

This is possible with the code locks that Java implements. It ensures that several threads can run at the same time, but never use a similar call to data that can create deadlock situations. It is simply implementing with the use of `synchronized` keyword in Java. There are various benefits of using this technique in concurrency programming in the language:

1. It ensures that a singular code block is only employed by a single thread in the same timeslot. This ensures that the duplication and overwriting of data is not possible.
2. Each thread is able to view the previous modifications that occurred to the data objects in the code. This ensures that no incorrect data is processed, allowing for the removal of deadlocks and inaccurate situations.
3. It provides the blocks of mutually exclusive access to code elements. This is important as it allows threads to communicate with each other and always ensure that the shared data is current with the needs of the concurrency in Java programs.

The keyword can be easily used when defining any method in Java. This ensures that only a single thread will be able to use the code block then, during parallel execution of different program threads. The next thread that needs this code will wait until the first thread has used and left the locked method. Here is an example of its use:

```
public synchronized void myMethod()
{
    // thread critical information
    // the intended functionality
}
```

The synchronization can also be used with individual code elements that may actually be a part of a method within an object. This then creates a locked block, which is guarded by a key. The key can be created in the form of an object or a string value to create the intended lock, required for ensuring that no problems occur during concurrency. Here is an example that helps present the use of locking code sections and blocks for seamless concurrency:

```
package java11.fundamentals.chapter19;
import java.util.ArrayList;
import java.util.List;
public class SynchronizationExample {
    private List<String> wishList = new ArrayList<String>();
    private List<String> shoppingCart = new ArrayList<String>();
    public void addToWishList(String product) {
        synchronized (this) {
            if (!wishList.contains(product)) {
                wishList.add(product);
            }
        }
    }
    /**
     *
     * Now the code moves to add the next product to the shopping cart.
     * If there are no products left in the wish list, it returns Null.
     */
    public String addToShoppingCart() {
        if (wishList.size() == 0) {
            return null;
        }
        synchronized (this) {
            // Checking if any product is available in the wish list
            if (wishList.size() > 0) {
                String s = wishList.get(0);
                wishList.remove(0);
                shoppingCart.add(s);
                return s;
            }
            return null;
        }
    }
}
```

Java also provides another method to ensure that threads do not make a mistake when picking up values for the required fields. This is accomplished by using the *volatile* keyword in the declaration of the variable for any Java class. It guarantees that whenever a thread is using this attribute, it will read the most recent value for the required information. However, it does not create an exclusive lock on the variable. There are conditions where this mode of functionality is required in a program.

A volatile variable automatically updates the variables as well, if they are modified within a single thread. This means that such a variable can often work as a reference for multiple values that may get changed during a temporary case scenario. The

setter is employed in such settings to initialize and assign a value to the variable. This allows for placing an address change and allowing the stored values to be available for other threads that may attempt to use it.

Concurrency significantly depends on the accurate use of the available memory for the JDK and the JRE. The memory model controls the communication between the memory which is assigned to the individual threads and the main memory available to the entire Java program, which is allocated by the JVM when the program is running on a particular computer.

The memory model is responsible for creating and defining the rules that govern the use of memory by different threads. It also controls the way information is communicated between the different threads. It describes the solutions for keeping memory available for the program. This is produced by allowing a thread to update its use of memory from the available main memory.

Atomic operations are important in Java, as they are defined as standalone tasks that must be completed without any interference from other program tasks. It is important that atomic operations are identified during their execution. Java allows the specification of a variable, when it is running an atomic state. This is automatically possible, but operations with long or double literals must be defined as atomic by using the *volatile* keyword with their declaration.

Normal operations like increments and decrements using the `++` and `--` operators are not considered atomic by Java, when performed on integer datatypes. However, by defining the value as an atomic one, it is possible to use such functionality. Java 9 onwards, Java supports the use of atomic variables that are already defined and have the necessary methods to perform different mathematical operations.

The synchronization is carefully maintained by the memory model as it updates information when it finds that a block of code is performing modification on a locked set. All previous modifications are available to the model that ensures excellent integration and the ability to simply remove all deadlock instances.

The simplest way to avoid problems with concurrency is to share only immutable data between threads. Immutable data is data which cannot be changed.

19.5.3 Concurrency Support

Java understands the importance of supporting concurrency. It provides several support methods that ensure that a class cannot be changed by proving several elements. All the fields and the class declarations are immutable when they remain finalized and their reference does not move during the construction of the class.

All fields that describe movable data objects remain private and are not used with a setter method. They are also not returned directly. Even if their value is changed within the class, it does not affect their use outside of the class in any manner whatsoever. This means that it is possible to have a fixed class that contains objects that have mutable data.

Take the example of mutable information such as Arrays that may be actually handed to a class externally, when the class is in its construction phase. The class can protect such elements by creating defensive copies of the required elements. This ensures that an object outside of the class cannot change the data in such fields.

Defensive copies play an important role in protecting your classes. This is important because other code elements can call for the class and in turn, change the data present in it in a manner not anticipated in your program's logic. A defensive copy is an excellent mechanism for ensuring data integrity. It works in a simple manner. Whenever a calling code asks for information, the immutable class creates a data copy and provides it to the code, without ever affecting the actual information stored in the immutable class. Here is an example of a defensive copy:

```
package java11.fundamentals.chapter19;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class DefensiveCopyExample {
    List<String> myArrayList = new ArrayList<String>();
    public void add(String s) {
        myArrayList.add(s);
    }
    // Following code creates a defensive copy and return the same. In this case, the
    // returning list remains immutable.
    public List<String> getMyArrayList() {
        return Collections.unmodifiableList(myArrayList);
    }
}
```

This program creates a new list for as a copy of the original list and passes it on to the relevant code. The original list never changes its status as array values.

Another important concept is a thread pool that holds the work queue for a Java program.

The pool keeps a record of all Runnable objects and constantly updates the list as and when required by the program. You can use the `execute(Runnable r)` code if you want the current thread to enter the queue where it will be called according to its preference that was explained earlier. The pool is named Executor and its implementation is utilized from `java.util.concurrent.Executor` interface. Worker threads can be easily added to the Executor, while there are also methods available to shut it down and terminate the thread processing.

Java can handle all types of asynchronous operations with the availability of this particular interface. There are different ways to implement execution tasks, such as using the Future interface. It allows asking for the results from a Callable task in Java. The CompleteableFuture option is available since Java 9 is very strong, as it ensures that all time-consuming activities are arranged in an ideal manner.

It allows the use of two approaches to provide concurrency in Java programs. The first is to ensure that all application blocks are arranged in a logical manner and follow the steps that are required to complete particular tasks. Another is to create a non-blocking approach where the application logic only moves with the flow of the tasks that are required for a program. This functionality allows the creation of different steps and stages and provides better control over the code callbacks that are required in any Java application.

19.5.4 Concurrency API Improvements

Java 9 contains considerable concurrency API improvements that occur as defined in the JEP 266 document. Here, we describe the improvements that are shared by the Java 9 documentation regarding the ability to better use `CompletableFuture` class and providing a lot more power in the language when compared with the older version.

The main focus of the concurrency improvements is the `CompletableFuture` API. The motivation behind these improvement steps is that the concepts of concurrency and parallelism must be fully integrated in the programming and execution requirement to give optimum control to a Java developer.

The improvements appear by providing better support in the relevant Java libraries, in turn adding the added functionality to the classes and the methods that may come under them, especially related to threads and concurrency settings. The interface improvements are also based on creating Reactive Streams that use the principles of concurrency and parallel thread processing.

Reactive Streams can provide better support through the use of the class `Flow`. It allows publishers to create items for different subscribers. The individual solutions can all use simple communication with the use of an information flow control to provide the ability to react to a dynamic program execution environment. A utility class is also added in the form of `SubmissionPublisher` that allows developers to create customized components capable of independent communication.

The main enhancement remains on the `CompletableFuture` API that was already discussed. Time-specific enhancements were certainly required during concurrency operations in Java as they had the ability to ensure consistency and make sure that all deadlock situations can be eliminated with accuracy.

There are excellent methods that are added to control the duration of different threads and their relevant operations. An Executor is added, which is extremely powerful and provides the use of a static method, such as `delayedExecutor`. It is a powerful method with the capability of providing a time gap between the reading of the code and the execution of a particular task.

When combined with the Future functionality, it can create excellent support for complete threads that can use concurrency, but still provide the reactivity necessary for enjoying the ideal support for improved parallel processing. There are several small improvements as well, which may not be apparent to a normal programmer, but significantly improve the capacity of handling multithreading that ensures reactive programming. This is built within the concurrency structure to offer improved throughput while maintaining greater control over enhanced heap sizes and shared memory spaces.

The overall concurrency improvements create a significant difference between the reactive programming capability of Java 8 and Java 9. The new programming kit employs the Flow API at its maximum power by providing four static interfaces. These interfaces contain all the methods that can provide a flow in program executions, where the publisher can control the production of data objects according to their consumption by different consumers of the program.

The production is handled by a Publisher, which produces the data required and received by different subscribers. The Subscriber acts as a receiver, where its argument depends on the definition presented by Subscription, that links it with the Publisher. Both parties combine together to create the Processor, which specifies a specific transformation requirement.

The data streams can be set up with the use of the Publisher by using the `subscribe()` method. It is used to connect a Subscriber with a Publisher. If a Subscriber is registered already, then this method registers an error and returns an illegal exception.

A Subscriber works to provide callback code for data items through the Flow library. It can be declared with `onSubscribe()` method. However, the same declaration can also occur with `onError(Throws exception)`, `onComplete()`, and `onNext(item)`. The first method describes a registration from allowing the requests for new data items to be moved to the subscriber. This situation is required for implementing the program flow measures that provide concurrent behavior.

There are also new methods that Java 9 update brings in the `CompletableFuture`. They allow the creation of different stages when performing concurrency functions. This ensures that the program always remains protected from sudden failures. A method file `completedStage()` and `failedStage()` are perfect for providing information that are in the processing phase. They can throw exceptions and provide information that may be useful for showing that a particular function is completed, ensuring the use of aggressive concurrency practices in the program.

The `CompletableFuture` is now more powerful, especially due to the support of delay and timeouts which are ideal for use in a large and complex program that often employs multiple threads and thread groups. The delay is an excellent choice for use in concurrent applications, where it simply associates the required time that significantly covers a thread from corrupting the available data values.

There is another method called `Timeout()`. It is excellent for ensuring that a particular future situation is eliminated before a relevant code must run. This situation may not always be required and therefore, the method only throws an exception if it is required for a `CompletableFuture<>` functionality. However, this functionality can be further improved by experimenting around and creating different logical approaches to achieve the desired functions.

Here is an important example of how the delay can be presented using different Flow controls:

```
package java11.fundamentals.chapter19;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
public class DelayedExecutorExample {
    public static void main(String[] args) throws InterruptedException,
ExecutionException {
        CompletableFuture<String> completableFuture = new CompletableFuture<>();
        completableFuture.completeAsync(() -> {
            try {
                System.out.println("CompletableFuture - Executing the code block");
                return "CompletableFuture completeAsync executed successfully";
            } catch (Throwable e) {
                return "In catch block";
            }
        }, CompletableFuture.delayedExecutor(3, TimeUnit.SECONDS))
            .thenAccept(result -> System.out.println("In Accept: " + result));
        for (int i = 1; i <= 10; i++) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Executing For Loop Block : " + i + " s");
        }
    }
}
```

The above program produces the following output.

```

CompletableFuture - Executing the code block
In Accept: CompletableFuture completeAsync executed successfully
Executing For Loop Block : 1 s
Executing For Loop Block : 2 s
Executing For Loop Block : 3 s
Executing For Loop Block : 4 s
Executing For Loop Block : 5 s
Executing For Loop Block : 6 s
Executing For Loop Block : 7 s
Executing For Loop Block : 8 s
Executing For Loop Block : 9 s
Executing For Loop Block : 10 s

```

This is a simple program that will present the running outside three times with values of 1, 2, and 3 seconds. The inside future will represent the screen output of processing data while presenting the required acceptance. The delay will then be overproducing the remaining 4, 5, 6, 7, 8, 9, and 10 seconds' values.

Methods that were not used in the previous version are removed and new methods are added in Java 9, which provide excellent support to the ever-improving concurrency API present in the programming environment. It includes improvement in the atomic functions that often employ reference array methods and Boolean comparison functions.

One problem that may appear with the Flow API is that sometimes, Publishers can produce data at a much faster rate, compared to its consumption by multiple Subscribers. These situations require the creation of an ample buffer, which can hold unprocessed elements. These elements produce backpressure, which Java 9 handles with the creation of logic that removes the collection of elements with various reactive programming techniques.

Reactive programming, when introduced with multithreading, truly empowers Java 9 and allows developers to use it at the best of their capacity. The development environment is still evolving to produce more API improvements. With the main improvement available in the Java package, it is inevitable that different developers may end producing better API support models that will help in using reactive programming principles with improved concurrency.

A conclusion to the concurrency API improvements focuses on describing how the use of effective evolution is required for using stronger techniques and methods such as multithreading. With improvement over the control of functions that are related to one another, it is possible to use the full advantages of concurrency as governed by its theoretical limit.

With the tools of CompletableFuture and new improvements like ForkJoinPool and other relevant classes, it is important to understand the individual threads in Java and how they can be set up to ensure the best use of multithreading and the ideal parallel processing with the help of parallel programming principles.

There are other API improvements as well. One important class in this regard is the ConcurrentHashMap, which is designed to support a system of concurrency retrievals. It is an excellent design capable of providing the Hash table functionality in an improved manner. It has the same methods, but all with improved performance. It is a class that does not employ locking, but offers all functions to remain safe from thread deadlock issues and other parallel processing problems.

The mapping class works well with retrieval and ideally allows the use of updating within the same processing zone due to not locking the code or data. Different parameters of the hash table can also be retrieved, allowing the use of enumeration or the iterator. The hash table has improved control and can also be resized if it faces a certain load. Size estimates are possible as well, by using the initialCapacity constructor. It is certainly a class that allows for improved concurrency in Java 9 when compared with the older JDK environments.

The newKeySet() method is available for setting the mapped values or simply recording the different positions. There are few differences from other classes, as it does not allow the use of null value. Concurrent hash maps are excellent for use when combining serial and parallel operations. They offer safe way to apply concurrency, while still ensuring that the ideal arguments can be used in the Java program.

ConcurrentHashMap generates a frequency map as well where it can produce a histogram of usable values. This means that it can support functions both in an arranged manner, while still containing a set of parallel executions that may be controlled with a single forEach action. Remember, the elements should always be used in a manner as to not get affected if the order of the supplied functions is changed since it will remain random, during bulk operations.

There are other operations of mapped reductions and plain reductions in this class. Plain reductions do not correspond to a return type, while mapped reductions are used for accumulating the application of functions. Sequential methods occur if the

map returns a lower size than the given threshold. The use of `Long.MAX_VALUE` provides suppression control on parallelism, while the use of 1 acts as allowing for maximum parallel results. This is possible because the program creates subtasks by using the `ForkJoinPool.commonPool()` method that allows parallel computations. The ideal programming starts by picking one of these extremes and then revising the values to achieve your required level of overheads against the delivered throughput.

The hash map can speed up the executions using parallel processing, but it is not always the preferred solution when compared to sequential processing. If small functions that are placed on separate maps are used, they will often execute slower than serial transformations. Parallel processing will also not be valuable if it is simply taking care of different tasks that are not related to each other, and do not produce a net gain over the normal capacity of the Java compiler.

The `ConcurrentHashMap` can also be created in the image of another map. This is an excellent option when you are experimenting with their use and have not yet identified the ideal approach to use the available `HashMap` options.

There are some excellent parallelism options since Java 9, such as the ability to use the `newWorkStealingPool()` method. This is a method present in the `Executor` class and allows the creation of a threading pool. It uses the available processes as the value for parallelism, and this defines the use of a process where all applicable processors are working simultaneously on different tasks. If the parallelism value is inserted in this method, then the thread pool keeps the required number of threads for the parallelism. The class then uses the creation of different queues to ensure that multiple threads are not in contention for the limited execution slots.

19.5.5 Dealing with Individual Threads

Let us once again discuss the individual threads produced by the JVM. Normally, the `main()` method creates a single thread on JVM. The thread continues to perform until the `exit` method of runtime class is called. The program ends when all non-daemon threads have performed their functions and have already been recalled. Another way for individual thread elimination is through an exception.

Individual threads can have several important parameters. It includes the name, the target, and the stack size available to the thread class object. The creation of individual threads is required especially when you want to implement the strong power associated with multithreading capacity of Java compiler.

There is excellent control available over the individual threads, with the `currentThread()` method allowing you to reference the currently processed object. There are other methods available as well, such as `yield()` which ensures that the thread is willing to drop its current use of processing for any other thread that is present in the processing scheduler.

The individual threads can also be controlled by momentarily making the caller unavailable for processing. Invoking a method like `onSpinWait()` is ideal for situations where you may want a loop construct that needs to show that the calling thread is, in fact, busy waiting for information from other parts of the program. This method keeps spinning unless controlled by a particular process or exception produced using the available flags. Here is an example:

```
package java11.fundamentals.chapter19;
public class OnSpinWaitExample {

    volatile boolean notificationAlert = true;

    public static void main(String args[]) {
        OnSpinWaitExample onSpinObj = new OnSpinWaitExample();
        onSpinObj.waitForEventAndHandleIt();
    }

    public void waitForEventAndHandleIt() {
        while ( notificationAlert ) {
            java.lang.Thread.onSpinWait();
            System.out.println("In While Loop");
        }
        processEvent();
    }

    public void processEvent() {
        System.out.println("In Process Event");
    }
}
```

The above example produces the following result.

The control over the individual threads is still available if they are grouped together. You can get the count of the threads, as well as learn about currently active threads. However, remember that it is possible to perform multithreading by allowing them to process in a concurrent manner, where they do not cause an interruption because of the shared memory space.

Another important concept to understand about individual threads is their ability to perform communication, which is termed as co-operation in Java. It is a method of allowing one thread to pause while ensuring that another thread can be executed in a particular order. It is accomplished using `wait()`, `notify()`, and `notifyAll()` methods that belong to the Object class in Java.

Let us start with `wait()` as it is a method that causes the currently operational thread to release its data lock and go into the waiting mode. The waiting can be for a defined period or only returned with the use of the `notify` methods. A time period can be mentioned for the return as the method argument or left without use for the following methods.

1. **notify()**: This method causes a single thread to come out of the waiting stage and become active on the current object. If there are multiple threads that are present within a single object, the selection of the awakening thread is random and lies at the discretion of the system components.
 2. **notifyAll()**: This method is excellent when you want all threads that are present in a particular object to come out of their waiting stage. This is important since it removes the random nature of the previous method and allows for better concurrency functions.

Remember, the `wait()` method is different from `sleep()`. The former method applies on the `Object` class while the latter works on the `Thread` class objects. The `wait()` method remains a non-static method and can be revoked with the use of `notify()` or `notifyAll()` methods, as well as specific time. On the other hand, `sleep()` works with a specific amount of time, and does not provide the dynamic control which is required when actively performing aggressive reactive programming that takes full advantage of the capabilities of the Java compiler.

19.5.6 Synchronizing Code Blocks

Synchronizing code blocks is an excellent way of creating particular methods and then controlling them to work in an organized manner. Synchronization can also occur in a limited capacity, where we can place some part of a method within a synchronized block while leaving the rest of it for random execution.

It is in fact, a way of locking parts of code that you do not want to be accessible to any other resource. Remember, the scope of using a synchronized block is always smaller than a complete method and should always be employed in this manner. Here is an example of a synchronized block used within a Java program:

```
package java11.fundamentals.chapter19;
public class SynchronizationBlockExample {
    public static void main(String args[]) {
        Calculator calculator = new Calculator();
        WorkerThread1 t1 = new WorkerThread1(calculator);
        WorkerThread2 t2 = new WorkerThread2(calculator);
        t1.start();
        t2.start();
    }
}
class Calculator {
    void multiplicationTable(int n) {
        // Following block will ensure the method is accessible in synchronized manner
        synchronized (this) {
            for (int i = 1; i <= 10; i++) {
                System.out.println(Thread.currentThread().getName() + " : " + n * i);
                try {
                    Thread.sleep(400);
                } catch (Exception e) {
                    System.out.println(e);
                }
            }
        }
    }
}
class WorkerThread1 extends Thread {
    Calculator t;
    WorkerThread1(Calculator t) {
        this.t = t;
        this.setName("Worker Thread 1");
    }
    public void run() {
        t.multiplicationTable(3);
    }
}
class WorkerThread2 extends Thread {
    Calculator t;
    WorkerThread2(Calculator t) {
        this.t = t;
        this.setName("Worker Thread 2");
    }
    public void run() {
        t.multiplicationTable(40);
    }
}
```

The above program produces the following result.

```

Worker Thread 1 : 3
Worker Thread 1 : 6
Worker Thread 1 : 9
Worker Thread 1 : 12
Worker Thread 1 : 15
Worker Thread 1 : 18
Worker Thread 1 : 21
Worker Thread 1 : 24
Worker Thread 1 : 27
Worker Thread 1 : 30
Worker Thread 2 : 40
Worker Thread 2 : 80
Worker Thread 2 : 120
Worker Thread 2 : 160
Worker Thread 2 : 200
Worker Thread 2 : 240
Worker Thread 2 : 280
Worker Thread 2 : 320
Worker Thread 2 : 360
Worker Thread 2 : 400

```

This program uses a synchronized section within a single method of `multiplicationTable()` which includes a counter that then produces a sleep delay for the thread. This situation results in generating a set of numbers where multiples of 3 are printed 10 times, while the same is then repeated with multiples of 40. Similar functionality can be obtained by using an anonymous class where you do not have to define it separately for the program operations.

There are static methods as well, which only use fixed information. Using synchronization for such methods ends up locking the entire class, rather than a particular object that calls for the method. Take the example of two objects from the same class that share information.

They may be termed as `ob1` and `ob2`. The use of synchronized methods will ensure that interference is not possible between different actions. The use of static synchronization is excellent because it ensures that the lock is available for the class and away from the individual objects. The lock is easily created on the class by using this method:

```

static void multiplicationTable(int n) {
    // Following block will ensure the method is accessible in synchronized manner
    synchronized (this) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(Thread.currentThread().getName() + " : " + n * i);
            try {
                Thread.sleep(400);
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }
}

```

This is the declaration which can be available on the class when used with the static keyword at the top with a defined class method.

19.6 | Understanding Deadlock

Java 9 update brings excellent multithreading support, which certainly brings deadlock into the discussion. Remember, a deadlock is an unavoidable reality when performing multithreading. It is the inevitable result of a situation where threads must wait for locked objects and work according to the defined thread rules. Figure 19.5 shows a visual representation of deadlock.

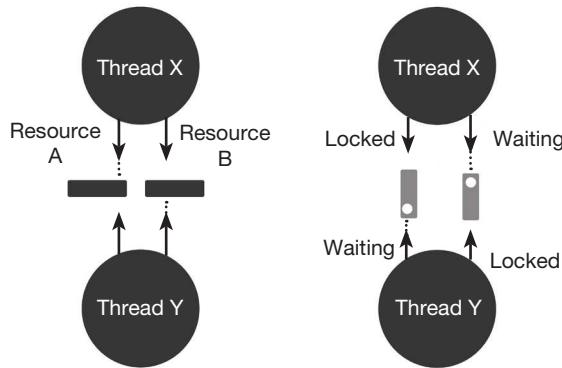


Figure 19.5 Visual representation of deadlock.

The deadlock is a natural situation that occurs in multithreading, when multiple threads are waiting for each other to release the lock on the object that needs to be processed, as the following example elaborates:

```

package java11.fundamentals.chapter19;
public class DeadlockExample {
    public static void main(String[] args) {
        final String firstResource = "First Resource";
        final String secondResource = "Second Resource";
        // Following code demonstrates thread 1 attempt to lock firstResource then
        secondResource
        Thread thread1 = new Thread("First Thread") {
            public void run() {
                synchronized (firstResource) {
                    System.out.println(this.getName() + " : First Resource is Locked");
                    try {
                        Thread.sleep(100);
                    } catch (Exception e) {
                    }
                synchronized (secondResource) {
                    System.out.println("Second Resource is Locked");
                }
            }
        };
        // Following code demonstrates thread 2 attempt to lock secondResource then
        firstResource
        Thread thread2 = new Thread("Second Thread") {
            public void run() {
                synchronized (secondResource) {
                    System.out.println(this.getName() + " : Second Resource is Locked");
                    try {
                        Thread.sleep(100);
                    } catch (Exception e) {
                    }
                synchronized (firstResource) {
                    System.out.println("First Resource is Locked");
                }
            }
        };
        thread1.start();
        thread2.start();
    }
}

```

The above code produces the following result.

Second Thread : Second Resource is Locked
First Thread : First Resource is Locked

This is an excellent demonstration of the deadlock situation. The first synchronization in thread 1 already sets first resource to locked, which is then suspended by using sleep. This means that the program proceeds to run thread 2, which produces the screen output about second resource, while once again sleeping to allow thread 1 to run.

However, the program is now stuck due to deadlock because thread 1 must wait for second resource to be free of lock in thread 2, which means that the next thread for processing is thread 2. The thread 2 faces a similar situation as first resource is similarly locked by the thread 1, creating a condition where the program is logically stuck.

19.6.1 Resolving Deadlock

Many programmers choose to simply ignore that it is possible to have a deadlock condition during the execution of a Java program. They believe that a deadlock may only appear due to a poor programming effort, and should always be removed during the planning phase of the program. They also believe since a deadlock can freeze a program, a simple solution would be to restart the application, resulting in a new state where the same deadlock condition may never appear again.

Obviously, this is the wrong way to go about Java programming as you must ensure from your end that a program is free from any conditions that may produce a stall or freezing, especially if it is providing control and support for a large-scale operation. Another way to go about it is to build a detection system in your program.

This is achieved by adding a special task that gets executed only to check the current status of the program parameters. This ensures that it is possible to detect, if the program is entering a deadlock situation, due to the reasons that we just described above. This is possible by checking whether tasks are stuck in their current functionality. The remedy can occur by eliminating a stuck task or forcefully liberating a shared resource, which is required for other program elements to function normally.

Another method is to produce a prevention of the Coffman conditions. These are the four ways in which a deadlock can occur during the program execution. A program can prevent a condition where special measures are built within the program structure that stop the occurrence of these four situations.

Another method to avoid deadlock situation is to make sure that your program design avoids deadlocks. This is possible by ensuring that your program first obtains the information about the required shared sources each time a particular task starts in your program. This ensures that there is always a set of available resources that allow your task to get executed without any problem. If there is a lack of resources, you can insert conditional delays that will always ensure that only those starts are initiated that can be completed in the current set of available resources.

19.7 | Concurrent Data Structures



Java 9 update brings excellent data structure options, but they are never designed to provide the ideal support for concurrent operations. The use of an external method ensures that ideal synchronization is possible. However, it significantly increases the computing time of your application. There are certain data structure practices that you can employ that will allow you to create data structures that are fully supported by Concurrency API. There are two groups of these structures – blocking and non-blocking data structures.

The blocking structures are present as methods that ensure that the calling task is blocked when you are employing the data structure although it does not have the intended values. On the other hand, non-blocking structures do not block the calling tasks. These methods are designed to throw an exception if a task calls on the data structure when it does not hold value. It can also produce a null value as a result.

19.8 | Multithreading Examples

Multithreading Java examples can be best followed by understanding the thread pool in Java. It is a collection of Runnable objects. Worker threads in it are controlled by the Executor framework and create a life cycle where it maintains the overall life of the Java program. It is possible to use:

```
Executors.newSingleThreadExecutor()
```

This is a method that will create a single thread, but one which contains multiple running items. Here is how it can be applied:

```
package java11.fundamentals.chapter19;
public class RunnableExample implements Runnable {
    private final long counter;
    RunnableExample(long counter) {
        this.counter = counter;
    }
    @Override
    public void run() {
        long total = 0;
        for (long i = 1; i < counter; i++) {
            total += i;
        }
        System.out.println(total);
    }
}
```

Once the thread is established, it can be executed in numerous ways to gain the required facilities. We have significantly discussed the concept of Future. Now we describe that the Callable object can also be employed with concurrent processing. When this is used with an executor, it returns an object of the Future type. The `get()` method can be employed to receive a Future object which can then be employed for designing the ideal concurrent processing schemes.

An excellent use of thread control is possible when it is mixed with the ideal use of the `sleep()` method. Defining subclass is another effective method when implementing new threads. These are instances where you can control individual thread instances to gain better performance.

Another key way in which multithreading is performed is by using the Swing application. This is created through a set of conditions where there is an initial thread that includes the `main()` method, which is designed to exit at the occurrence of a defined event. This situation then gives rise to another event dispatching thread (EDT), which runs the specific functionality required from the program.

This technique provides excellent control over a program which may perform better with concurrent behavior. Multithreading is excellent when it can be divided in the form of different tasks that can then be set up according to the occurrence of key events during program execution. The main program then becomes a controller which keeps the operations shifting to the required threads. Remember, there is always the need to have a background (daemon) thread available, which takes care of the intensive tasks and the input/output controls that you need in a program environment.

When threads are switched according to their functionality, it is possible that application users may identify a pause between their inputs and the response produced by the application. This means that the EDT, which is set up in the program must never work for over 100 milliseconds, where it may start to produce a noticeable delay in the functioning of the worker threads.

Multithreading problems can be avoided in Java programs when the problems that are required for controlling the graphical user interface (GUI) are also performed on the EDT. This ensures that all operations remain thread safe, and can be carried out without ever causing any problems.

On the other hand, tasks that include the input/output processes should be kept on the main thread as they need to provide swift interaction, otherwise the program behavior is slow and does not produce the intended benefits. Another capacity for use is that Swing operations can be updated with the use of a delay timer. This allows the thread to update its components at controlled time intervals, resulting in the generation of the required information.

Another capability is with the publishing of thread information. This can happen with the use of:

```
protected final void publish(V... chunks)
```

The above method, which should be called inside the `doInBackground()` method, sends data to the process method to deliver intermediate results

```
protected void process(List<V> chunks)
```

The above method works in an asynchronous manner to receive datasets from the `publish()` method.

This is a program that produces the intermediate results. The method can be useful for implementing improved controls in the program. It allows Java programs to produce the situation of the current tasks. This allows for the implementation of practices that result in a better control over the data streams that are available in the program.

19.8.1 Matrix Multiplication

A common problem which you can use to learn concurrent programming is to create a matrix multiplication program. You can learn the simple concept of matrices where it is possible to multiply two matrices, as long as the number of columns of the first matrix is equal to the number of rows of the second matrix. This is termed as having matrices $A(p \times q)$ and $B(q \times r)$ where the first value in the parentheses depicts rows and the second value describes columns.

This operation is possible using different schemes. The Java library recognizes this need as an important function and provides the `MatrixGeneratorExample` class for these operations. This class uses the following code for its operation:

```
package java11.fundamentals.chapter19;
import java.util.Arrays;
import java.util.Random;
public class MatrixGeneratorExample {

    public static void main(String args[]) {
        System.out.println(Arrays.deepToString(generateMatrix(3, 3)));
    }

    public static int[][] generateMatrix(int rows, int columns) {
        int[][] matrix = new int[rows][columns];
        Random random = new Random();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                matrix[i][j] = random.nextInt() * 10;
            }
        }
        return matrix;
    }
}
```

The above program produces the following result:

```
[[853339708, 1266531654, -21594734], [1024418966, -490434492, -2020854294], [1604225446, -2030454084, 568464094]]
```

As we can understand from the structure of this class, it is easy to perform this calculation using a serial method. This will be a method where we will use two matrices and then use loops to multiply the elements that are present in the rows and columns

of the matrices. The results will be updated each time when calculating the value, and the loop will end when all elements are multiplied.

You should always produce a sequential version of every algorithm before you employ parallel versions. Here is the sequential method for multiplying matrices:

```
package java11.fundamentals.chapter19;
import java.util.Arrays;
import java.util.Random;
public class MatrixSerialMultiplierExample {

    public static void main(String args[]) {
        int[][] firstMatrix = generateMatrix(3,3);
        int[][] secondMatrix = generateMatrix(3,3);

        System.out.println(Arrays.deepToString(multiplyMatrix(firstMatrix, secondMatrix)));
    }

    public static int[][] generateMatrix(int rows, int columns) {
        int[][] matrix = new int[rows][columns];
        Random random = new Random();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                matrix[i][j] = random.nextInt() * 10;
            }
        }
        return matrix;
    }

    public static int[][] multiplyMatrix(int[][] matrix1, int[][] matrix2) {

        int row1 = matrix1.length;
        int column1 = matrix1[0].length;
        int column2 = matrix2[0].length;
        int[][] result = new int[row1][column1];

        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < column2; j++) {
                result[i][j] = 0;
                for (int k = 0; k < column1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }
        return result;
    }
}
```

The above program produces the following result.

[[-59142824, -631908344, -1314105512], [-341256416, 293358376, -1805709024], [194502204, -108105864, 1115015312]]
--

This is a serial program that uses three matrices. The first two matrices are multiplied using the basic matrix multiplication rules and the results are stored in the `result[][]` matrix. This program implies that both matrices are fit for performing the mathematical operation. This can be checked using other programming elements. We can also check the execution time by using a random generator to multiply large enough matrices, like ones with 2000 rows and columns, to get how many milliseconds it takes to reach the result. Here is a small section which shows how you can record time for the matrix multiplication:

```

package java11.fundamentals.chapter19;
import java.util.Arrays;
import java.util.Date;
import java.util.Random;
public class MatrixSerialMultiplierWithTimeExample {

    public static void main(String args[]) {
        int[][] firstMatrix = generateMatrix(3,3);
        int[][] secondMatrix = generateMatrix(3,3);

        Date start=new Date();

        System.out.println(Arrays.deepToString(multiplyMatrix(firstMatrix, secondMatrix)));

        Date end=new Date();

        System.out.printf("Total Time Taken : %d milli seconds", end.getTime() - start.
getTime() );
    }

    public static int[][] generateMatrix(int rows, int columns) {
        int[][] matrix = new int[rows][columns];
        Random random = new Random();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                matrix[i][j] = random.nextInt() * 10;
            }
        }
        return matrix;
    }

    public static int[][] multiplyMatrix(int[][] matrix1, int[][] matrix2) {

        int row1 = matrix1.length;
        int column1 = matrix1[0].length;
        int column2 = matrix2[0].length;
        int[][] result = new int[row1][column1];

        for (int i = 0; i < row1; i++) {
            for (int j = 0; j < column2; j++) {
                result[i][j] = 0;
                for (int k = 0; k < column1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }
        return result;
    }
}

```

The above program produces the following result.

```
[ [676643992, 1091729664, 1847180544], [1048256660, -2143242596, -332108212], [-1842184312, -1674877996, 974596884] ]
Total Time Taken : 1 milli seconds
```

This section of the code will first record the time before the start of the matrix multiplication and then also record the time when the final calculation has occurred. The output to the console is done by subtracting both times; this results in the number of milliseconds it took to complete the operation.

Now that we have learned how to carry out this program, it is time to prepare concurrent versions to see the difference. Sequential programming often has only one avenue, but there are various methods for parallel processing. You can use a single thread for each element when creating the result matrix. You can also employ an individual thread for every row for the same matrix. Another effective way of concurrent programming is to employ all the threads that are available in the current JVM structure.

The first method creates too many threads, as the number of elements are too great for large matrices. Take the example of 2000×2000 matrices. Multiplying these matrices will result in the form of a matrix that will require the calculation of 4,000,000 elements. Although this can be accomplished, we will not be executing this program in this book. However, we will represent the other two options here.

19.9 | Designing Concurrent Java Programs



There are several ways to create a concurrent Java program, one that employs the best principles of parallelism, while using the different resources and tools that are present, especially in the latest Java development environment. Here is a strategy that you can use to ensure that you always end up with the required concurrent algorithms for use, regardless of the actual functionality that you need in the program.

A good practice in this regard is to always create a sequential version of the algorithm that you want. This is important as it allows you to measure the advantages of using concurrent code, as well as understand whether both versions are producing the same program results, which is important in all applications.

The creation of an initial sequential program will allow Java programmers to later compare the throughput of both program versions. This will ensure that you have hard numbers that describe the results of the responsiveness of the program, as well as judge the amount of data that both algorithms processed, as the overall output is always the same.

1. The first step in creating the ideal concurrent algorithms is to analyze the parameters and the delivered performance of the sequential version of your program. Special attention must be given to sections that especially took a lot of time or used a significant portion of the available heap to the Java program. Some ideal options in this regard are the loops and decision-making elements in the program that can take a lot of time before producing the ideal results.

Other analytical elements include looking at independent parts of the program, which would not get affected, even when using concurrent programming. It may also include the object initializations and setting up the initial variables and data elements that are required for a Java program.

2. The second step in this process is to perform the concurrent designing phase. This is possible when you have analyzed your sequential code and can create a logical picture of what parts are independent and can be truly initiated with parallel program processing. There are two ways of designing concurrency. The first is to decompose the program in the form of independent tasks that can occur simultaneously. The second is the decompose the program in the form of independent data elements. You can create datasets that share resources, where you create protected access to ensure that all important data remains locked before a critical section is executed.

Always remember that the aim of your concurrent program is to create a situation where it is important to produce a benefit. If you find that adding more functional elements will create a situation where parallel execution will not be feasible, you can choose to split the program elements to make sure that the benefits are significantly available.

3. The third step to performing concurrent algorithms in Java is to carry out the implementation using the available thread, parallelism, and concurrency options. Java, with all the tools that we have mentioned in this article, is perfectly suited to provide this option using different thread classes and the ideal Java library tools.
4. The fourth step in this practice is to test your concurrency supporting code. The parallel algorithm must be tested against your sequential code, where you must compare the important results from both avenues. We will later describe how you can monitor concurrent Java applications for performing the ideal tests that deliver useful information. The testing phase will often include debugging as well, where you simply remove any programming errors.
5. This moves us to the fifth step of tuning your concurrent program. This step may not be always required, as you should only perform customized tuning if you do not get the intended benefits from your parallel processing practices. There are some important ratios that you can use when measuring the performance of concurrent applications, such as the speed up metric. This is a ratio between the execution time of the sequential program to the execution time of the concurrent program. It should always return a value greater than one, showing that there is a time benefit for using concurrent processing. The Gustafson's Law is employed when creating parallel designs that have the ability to use multiple cores for different input datasets. This is applied with the formula:

$$\text{Speedup} = N - P * (N - 1)$$

where P represents the percentage of the program code which can be parallel-processed without affecting the program integrity and N denotes the number of available cores which can all function with their separate datasets at the same time.

There are some other points that you should also keep in mind when using parallel processing with the ability to react to the real-time application needs. Remember, not all algorithms are empowered with the use of parallel processing. Programs where it is possible to run several independent threads at the same time are ideal for implementing concurrency and multiple threads for achieving the best operational performance.

There are some important points that you should always consider before using concurrent Java tools in your programs. The efficiency of your parallel program must be significantly greater than sequential processing. This is shown by either a smaller running time or the ability of the concurrent program to process more data in the same timeframe.

Your algorithm using reactive programming elements must still focus on simplicity. Remember, the final Java program should always have the simplest form, which allows for easy testing, maintenance and other programming steps, which are required before the application is ready for live use. Portability is also important where your parallel programming solution must provide the same benefits over a number of varying platforms.

The last important element is that your parallel program must have the factor of scalability. Your program should provide the same or even greater benefits when the number of available sources for the program are increased by a considerable margin. It should always be possible to scale up and create a global application from your specific code.

19.9.1 Ideal Tips for Concurrent Java Programs

Good concurrent programs are possible when you, as a Java developer, understand your core objective. This objective is to employ concurrency as a way to enhance the performance of your application, while ensuring that it offers significant throughput benefits. Remember, you can always learn about the code and the syntax of Java. Your learning focus should always be on understanding the benefits of employing concurrency in various Java application requirements.

Some standard practices can help ensure that you produce the optimal performance from the use of concurrency. We will describe some important concurrency tips and tricks in greater detail in the following subsections.

19.9.1.1 Never Assuming Thread Order

If you do not set up synchronization schemes, you can never be sure of the order of the execution of the different program threads in concurrent programming. The thread scheduler of the specific operating system decides which task is executed first if there are multiple available options, with no order specified with the use of localization.

Assuming this will never cause an issue leaves your program with conditions that give rise to data race conditions as well as thread deadlocks. There are several applications where the exact order of the tasks in the algorithm can affect the final result of the program. You cannot be sure of the reliability of your code in such a situation. With the right synchronization methods and schemes in place, you can make sure that your concurrent program always behaves in an intended manner.

19.9.1.2 Building Scalability Option

The main objective of your concurrent program is to take the maximum advantage of the available computer resources. This includes the available memory and the processing cores. However, computing elements may not remain the same as the supporting hardware is subject to an upgrade, especially if your client decides to scale its business project.

This means that you should build scalability in all your programs. Scalable programs are great as they provide a time proof design for your clients. Good Java programmers ensure that they never assume the number of available cores or heap sizes. The ideal program would always have a reactive programming element in it, which always finds out the available resources and then implement them in a maximum capacity in the different program functions.

This is possible with the method of `Runtime.getRuntime().availableProcessors()`, which returns the number of available processors. You can then set up your program to make use of this information in a reactive manner. This practice may increase your program overhead slightly, but it offers amazing scalability advantages, where your program can enjoy improved use of the available resources.

Designing scalability can be difficult if you perform concurrency with the use of task decomposition, rather than setting data decomposition schemes. The independent tasks can then mean that the overhead will be great because of the required

synchronization methods. The overall application performance will actually go down if your program needs to process the need for using the available processors. Always study whether you can create a dynamic algorithm which makes use of dynamic situations. Otherwise, only build limited scalability where the overhead should never exceed the advantages that can be gained from the use of additional processors.

19.9.1.3 Identifying Independent Code

The best performance from concurrency is only possible when you have identified all the independent tasks in your program. You should not run concurrency for tasks that depend heavily on each other, as this will simply require too much synchronization, negating the benefits of parallel processing. These tasks will run sequentially and the additional code will simply place a burden on the program. However, there are other instances where a particular task will depend on different prerequisite functions that are all independent of each other.

This situation is great for concurrency as you can run all the prerequisite execution in parallel and then place a condition for synchronization. You allow the task to initiate only when information about all the required functions has already been processed. Always remember that you cannot use concurrency in loops, as the next loop iteration often includes the use of data instances which are set up or worked on in the previous loop iteration.

19.9.2 High-Level Concurrency

The Java concurrency API provides various classes to perform the ideal parallelism in your programs. You can use the Lock or Thread classes, and you need to focus on the executors and fork/join facilities. This provides you access to concurrency at the highest level, like carrying out separate tasks with the use of multithreading. Here are the advantages of ensuring that concurrent tasks are specified at the highest level in your Java program:

1. The management of threads does not remain your responsibility when you allow Java to manage the intricate details of the required threads. The Flow API will manage these tasks and allow you to only create high-level functionality with a clean code.
2. Since this practice will ensure that the created threads are directly employed, they always occur in an optimized manner. The pool of threads is created according to the needs of the program once and then employed multiple times as and when required by your code. This allows you to automate several concurrency tasks.
3. The use of advanced features is possible with the use of thread groups and pools. The executors provide Future objects and ensure that you can use the main concept of concurrency for optimal benefits.
4. Your code is simple to execute in all JVM environments. The threading and other functions simply follow the available limits of the physical resources that they have available on different executing machines. This makes it easier to scale your operations and perform hardware migrations.
5. Your application is quicker to run, especially in environments that have access to the latest JVM and JRE versions. This is possible with continuous internal improvements and the ability to employ specific compiler optimizations.

19.9.2.1 Use of Immutable Objects

Another ideal practice is to avoid data race conditions in your concurrent programs. This is possible by creating immutable objects in Java. These objects are special because their attributes cannot be modified after their creation. This means that they cannot get altered during multiple executions that define the concurrency practice.

The String class in Java is an excellent example of this concept. It always generates a new object whenever mathematical operators are applied in the objects of this class. Immutable objects do not need to be synchronized since it is not possible to modify the important values in these classes. The modification of any object parameters simply gives rise to a new object, which means that the main attributes always remain fixed.

Another advantage of these objects is that you can always count on the consistency of the data that you have in your program, since class defining characteristics are always secure during program executions. However, if your program ends up creating too many objects, its performance can seriously go down because it will use greater memory and reduce the throughput. Complex objects that often contain other objects defined within them can create serious issues, and you must use this practice with care.

19.9.3 Controlling Sequential Executions

Sequential executions will always occur in a program. Your focus should always remain on reducing the amount of time you need to keep the other tasks locked up so the critical section of the program could perform its execution. Special attention is needed to ensure that you create locks and blocks that are time sensitive and ensure that your program does not suffer from an imbalance of serial and parallel tasks during execution.

This is a situation which is often simplified with the creation of high-level concurrency. This will allow you to create shorter critical sections and avoid situations where your library can quickly supply the required code during executions. A good example would be to use the available library documentation and make the best of the Java classes. One excellent code for this is the `compute()` method which is present in the `ConcurrentHashMap` class in Java.

You should also avoid placing blocking operations in your critical section. These are operations that can block the tasks that call them for use. These tasks are then only available after a particular instance is achieved, like reading a file or outputting data to the console. The performance of your program degrades because blocking operations are slow and may stop the rest of your program from operating until the results of these operations are produced.

19.9.4 Avoiding Lazy initialization

One trick for successful Java programming is to employ lazy initialization scheme. It is a method where you do not create an object before it is required for the first time in your program. This practice offers the advantage of reducing the load on the available heap for the Java program. You get on in your code, only creating objects that will be required in a specific execution.

However, the same situation can be a problem in concurrent programs, where you cannot control where an object may be required for the first time, especially with aggressive multithreading strategy. If you are using singleton classes, then it will not be possible to use singular objects. You can learn more about using the on-demand holder in Java by visiting the official guide on Oracle at <https://community.oracle.com/docs/DOC-918906>, which takes care of this problem and ensures that you can take advantage of lazy initialization as much as possible.

Summary

This chapter explains multiple themes that are related to the use of multithreading in the Java environment. We find that the use of multithreading brings in the application of parallel processing, which is often described in the form of concurrency in Java programming. There are several ways of implementing concurrency and designing code improvements that ensure that your Java program performs better than a typical sequential program.

We find that it is always the ideal practice to implement concurrency schemes at the highest level in your programs. In fact, with the availability of the worker thread pools and the functions offered by the executors, it is always better to leave the choice of selecting the threads and their concurrent behavior to the JVM itself. This will ensure that the ideal resources are employed each time the program operates, in various hardware environments.

We also share several important tips that help you avoid the common concurrency mistakes when programming in Java. We believe that if you focus on creating programs that use threads to match the available processors on the executing machines, it is possible to perform useful reactive programming. You can ensure that your program reads the available hardware resources before starting its working functions, and then make sure that it uses a pool of worker threads accordingly for maximum concurrency benefits.

There are times where you may never have to make difficult choices, especially with the ideal thread functionality offered by the Java Flow API. It certainly empowers concurrent programming and allows developers to focus on developing algorithms, which offer solutions to the client problems. The enhanced Java tools are sure to take care of the required thread, concurrency and reactive programming needs in each case in an ideal manner. Java is certainly an excellent choice for carrying out reactive programming, especially in various client/server applications.

In this chapter, we have learned the following concepts:

1. Reactive programming.
2. Multithreading and programming with multithreading.
3. Concurrency and advantages of concurrency
4. Concurrency API improvements.
5. Dealing with individual threads.

6. Synchronization blocks.
7. Understanding and resolving deadlocks.
8. Concurrent data structures.
9. Multithreading examples.
10. Designing concurrent Java programs.
11. Ideal tips for concurrent Java 11 programs.
12. High-level concurrency.
13. Controlling sequential executions.
14. Avoiding Lazy initialization.

In Chapter 20, we will explore the world of Spring and Hibernate. The chapter focuses on Spring Framework and Spring MVC. We will learn concepts like Inversion of Control and Dependency Injection. We will explore bean scopes like singleton and prototype. In addition, we will look into the role of controller and DispatcherServlet.

Multiple-Choice Questions

1. Which of the following is a type of multitasking?
 - (a) Thread based
 - (b) Process based
 - (c) Process and thread based
 - (d) None of the above
2. Which of the following is a thread priority in Java?
 - (a) Float
 - (b) Integer
 - (c) Double
 - (d) Long
3. Which of the following methods is utilized for launching a new thread?
 - (a) `run()`
4. Which of the following is the default priority of a thread?
 - (a) 1
 - (b) 5
 - (c) 0
 - (d) 10
5. You can use the same thread object to launch multiple threads.
 - (a) True
 - (b) False

Review Questions

1. What is multithreading?
2. How is multithreading useful?
3. How will you write a concurrent program?
4. What is thread life cycle?
5. How can you make a thread wait for other threads?
6. What is deadlock? How do we avoid it?
7. What is reactive programming?
8. What are concurrent data structures?
9. What are the improved concurrent APIs?
10. What are the advantages of concurrent programs?
11. How will you set thread priority to give preference to one thread over other?

Exercises

1. Write a program that accepts stock stream data about current stocks and their prices. Write multithreading code to manage the stream and process in different threads.
2. Write a program for an ATM machine that dispenses money. Make sure this program uses multithreading reactive programming concepts. Also note that someone may transfer money from online backing or from a branch at the same time a user is withdrawing cash from the ATM, so you have to make sure the cash withdrawal process is secure in a way that a user should not be able to withdraw more than he/she has in his/her account.
3. Write a program using reactive programming for an online gambling application. Ensure that multiple people cannot bet on an event. You have to make sure everything happens in a timely manner.

Project Idea

Create an airline ticket booking application. Any person should be able to book a flight ticket from an airline's website, ticketing counter, or through a travel agent. Please note that flight prices are not fixed, they fluctuate based on demand. Hence, price may change between checking the price and

booking a ticket. Also, tickets are limited per flight; thus, in a limited ticket availability case, the ticket may be booked by an agent while a user is trying to book it from the website. Hence, you have to make sure you use proper locks in case multiple threads are trying to book a ticket.

Recommended Readings

1. Javier Fernandez Gonzalez. 2017. *Mastering Concurrency Programming with Java 9*. Packt: Birmingham
2. Mayur Ramgir and Nick Samoylov. 2017. *Java 9 High Performance*. Packt: Birmingham
3. Paul Bakker and Sander Mak. 2017. *Java 9 Modularity: Patterns and Practices for Developing Maintainable Applications*. O'Reilly Media: Massachusetts

Introduction to Spring and Spring MVC

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- Spring Framework.
- Inversion of Control and Dependency Injection.
- Dependency injection variants.
- Bean Scopes such as Singleton and Prototype.
- Spring MVC.
- Role of controller and how MVC works.
- Role of Dispatcher Servlet.
- Concept of Dependency Injection and Inversion of Control.
- Maven and how to use it.
- How to connect database to perform CRUD operations.
- How to design a view to display data processed by Model through Controller.

20.1 | Spring Framework



In the concept of web application at a higher level, we have a presentation layer, a business layer, and a database layer. The days before Spring was invented, most of the applications were developed using the concept called *procedural programming* in which a specific task is done by calling the required libraries by the logic code. In other words, a task is achieved by performing a series of computational steps in a specific order. Spring is built on concepts called *inversion of control* and *dependency injection* (IoC and DI in short, respectively). It allows building applications from Plain Old Java Objects (POJOs) and apply other required services in a non-invasive manner to achieve the desired functionality.

QUICK CHALLENGE

Mention all the modules of the Spring framework.

Let us explore these topics in detail to understand the inner working of Spring.

20.1.1 Inversion of Control

As we have seen in the traditional programming model, the logic code makes calls to the required libraries to perform a series of tasks. IoC does the exact opposite, where the control remains with the framework which executes the logic code. This promotes loose coupling, which means the objects of the functional classes do not depend on the other required objects' concrete implementation.

IoC provides various benefits such as modularity, loose coupling, etc., where the functionalities of the program get divided into various areas. This allows loose coupling where the components are not dependent on each other's implementation. Hence, it offers flexibility to modify the implementation of these objects without affecting the other parts of the application. This also enables testing teams to test the functionalities in isolation and later test the dependable functionality by mocking the object dependencies.

IoC can be achieved with the help of DI, which we will explore now.

20.1.2 Dependency Injection

In a typical application, there is a need to work with a lot of individual objects in order to develop a required functionality. For example, for a school registration system, we will need a student object, teacher object, classroom object, subject object, etc. These objects should be accessible in a business logic class such as Registration.java, which will enable the user to register for courses. In traditional development, you will need to initialize all these objects in order to use them. In other words, you will create these objects by yourself and need to manage the life cycle of each object for the optimum use of resources such as memory. This is where DI comes handy. DI is a structural design pattern that eliminates the need for us to initialize these required objects and manage the life cycle by ourselves. It solves this problem by injecting the required object to the constructor or setter by passing as a parameter. Figure 20.1 shows the use of DI. This DI functionality is implemented in a library called *inversion of control* (IoC) container. This IoC container is responsible for injecting the dependent objects when the bean creation is taking place. Since this operation is inverse of the traditional approach, it is called inversion of control.

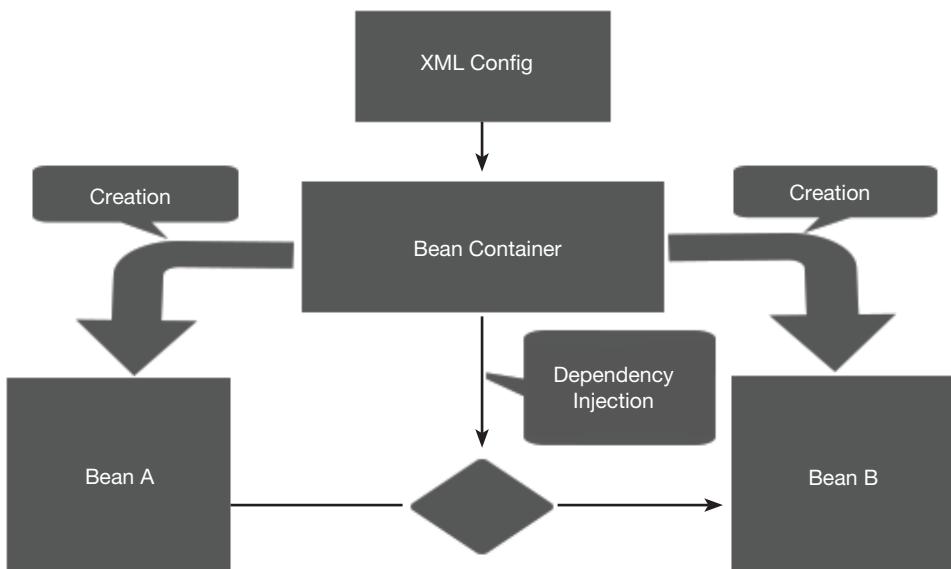


Figure 20.1 Representation of use of dependency injection.

Let us understand this better with an example. We will first take a look at the traditional approach, in which we will create an object dependency.

```

package java11.fundamentals.chapter20;
public class ObjectDependencyTraditional {
    private Product product;
    public ObjectDependencyTraditional() {
        product = new Product("My Awesome Product");
    }
}
class Product{
    private String name;

    public Product(String name) {
        this.name = name;
    }
}
  
```

In the above example, we have to instantiate the dependent object “product” within the ObjectDependencyTraditional class. Now, let us take a look at the DI approach.

```
package java11.fundamentals.chapter20;
public class ObjectDependencyWithDI {
    private ProductDI product;
    public ObjectDependencyWithDI(ProductDI product) {
        this.product = product;
    }
}
class ProductDI{
    private String name;

    public ProductDI(String name) {
        this.name = name;
    }
}
```

In the above example, IoC will inject the object of ProductDI while creating ObjectDependencyWithDI bean.

There are multiple ways you could use DI. Before exploring these, let us first understand how the IoC container works. IoC is mainly responsible for instantiating, configuring, assembling objects, and managing their life cycles. These objects are known as *beans*. Spring provides ApplicationContext interface to implement DI mechanism. This interface is a subinterface of BeanFactory interface which provides advanced configuration method to manage all types of objects. We can consider BeanFactory as the central registry which holds the configuration of all the application components. ApplicationContext extends it to add more enterprise-specific functionality. ApplicationContext interface has several implementations for various purposes; for standalone application it provides FileSystemXmlApplicationContext, for web applications it has WebApplicationContext and ClassPathXmlApplicationContext to provide a configuration for the context definition file.

Let us see how to instantiate a container to use to get the managed objects instances.

```
ApplicationContext appContext = new ClassPathXmlApplicationContext("MyApplicationContext.xml");
```

In the above example, we are using ClassPathXmlApplicationContext to get objects configurations from an XML file that is located on the class path, which is then used by the container to assemble beans at runtime.

Before we deep dive into variants of DIs, let us first explore the concept called *bean scope*. This is an important concept as it will help improve the application performance.

Flash



How do we perform dependency injection in Spring framework?

20.1.2.1 Bean Scope

In the above section, we have seen how IoC container using DI creates beans that are needed by the application. Although the IoC container creates these required beans for us, Spring provides a mechanism to have more granular control on the creation process by specifying the bean scope. The Spring framework provides five scopes. These are singleton, prototype, request, session, and global-session. Let us explore these scopes in detail:

1. **singleton:** The framework first looks for a cached instance of the bean and if it cannot find one, it creates a new one.
2. **prototype:** The framework will create a new bean instance on every request. In order to use the other three scopes listed below, you need to use a web-aware ApplicationContent.
3. **request:** This is similar to the prototype scope but only for web applications, where a new instance is created on every HTTP request.
4. **session:** As the name suggests, this scope is limited to every HTTP session. A new instance is available per session.
5. **global-session:** This scope is useful for portlet applications where a new instance is created for every global session.

Here is how you define the scope:

```
<bean id="ConstructorBasedDependencyInjectionSimpleTypeExample" class="java11.fundamentals.chapter20.ConstructorBasedDependencyInjectionSimpleTypeExample" scope="singleton" >
    <constructor-arg type="java.lang.Boolean" value="true"/>
    <constructor-arg type="java.lang.String" value="Mayur Ramgir"/>
</bean>
```

In the above example, `<bean>` definition provides an attribute called `scope` where we can specify the type of scope we need. If no “scope” attribute is provided, Spring considers the bean as “singleton”. In other words, singleton is the default bean scope.

There are multiple ways we could use DI in our application. In subsection 20.1.2.3, we will explore these ways in detail. However, before deep diving into this concept, we must first understand the concept called Autowiring. This is a mechanism used by Spring container to inject dependencies to the class.

20.1.2.2 Autowiring Dependencies

Spring container uses autowiring to automatically resolve dependencies. There are four modes by which we can autowire a bean in XML configuration.

1. **Default mode:** In this mode, no autowiring is used and we have to manually define the dependencies.
2. **byName mode:** In this mode, the autowiring is done using the name of the property. This name is used by Spring to look for a bean to inject.
3. **byType mode:** In this mode, the type of the property is used by Spring to look for a bean. If more than one beans are found, Spring throws an exception.
4. **Constructor mode:** In this mode, constructor of the class is used to autowire the dependencies. Spring will look for beans defined in the constructor arguments.



Are singleton beans threadsafe?

20.1.2.3 Variants of Dependency Injection

There are mainly three types of variants of DI, which can be used to set the dependencies for the class. These are constructor-based, setter-based, and field-based DI. Constructor-based DI is the most recommended way of injecting the required dependencies. It has many advantages over the other two methods. Setter-based injection is the next recommended way. It is highly discouraged to use field-based DI due to several disadvantages. Let us explore these in more detail to understand the reasons for these recommendations.

20.1.2.3.1 Constructor-Based Dependency Injection

This method is useful to inject the required dependencies via constructor arguments. Let us see the following example that shows a class in which dependencies are supplied via constructor arguments. Please note that this is a simple POJO class which does not have any dependencies on any of Spring container’s interfaces, classes, or annotations.

```
package java11.fundamentals.chapter20;
public class ConstructorBasedDependencyInjectionExample {
    private Tyre tyre;
    private HeadLights headlight;
    public ConstructorBasedDependencyInjectionExample(Tyre tyre, HeadLights headlight) {
        this.tyre = tyre;
        this.headlight = headlight;
    }
}
class Tyre {
    public Tyre() { }
}
class HeadLights {
    public HeadLights() {}
}
```

Constructor argument resolution: As we have seen in the example in Section 20.1.2.3.1, we are passing two parameters to the constructor `ConstructorBasedDependencyInjectionExample`. Since these two parameters are of different types and not related by inheritance, there is no ambiguity and hence it is easier to resolve the matching process. In this case, the order in which the parameters are supplied will be used to instantiate the bean. Hence, the following configuration will work without a need to explicitly specify indexes and/or types in the `<constructor-arg>`. Let us see the following example:

```
<beans>
<bean id="ConstructorBasedDependencyInjectionExample" class="javall.fundamentals.chapter20.ConstructorBasedDependencyInjectionExample">
    <constructor-arg ref="tyre"/>
    <constructor-arg ref="headlight"/>
</bean>
<bean id="tyre" class="javall.fundamentals.chapter20.Tyre"/>
<bean id="headlight" class="javall.fundamentals.chapter20.HeadLights"/>
</beans>
```

Constructor argument type matching: In the above case, we are using beans as parameters, hence the type is known. This helps in matching the parameters easily. But what if we use simple types like `<value>false</value>`? It is not possible for Spring to figure out by itself whether it is a Boolean or String. Let us see the following case:

```
package javall.fundamentals.chapter20;
public class ConstructorBasedDependencyInjectionSimpleTypeExample {
    private Boolean healthy;
    private String name;
    public ConstructorBasedDependencyInjectionSimpleTypeExample(String name, Boolean healthy) {
        this.name = name;
        this.healthy = healthy;
    }
}
```

In the above case, we need to explicitly mention the type in the configuration in order for Spring to resolve the arguments.

```
<beans>
<bean id="ConstructorBasedDependencyInjectionSimpleTypeExample" class="javall.fundamentals.chapter20.ConstructorBasedDependencyInjectionSimpleTypeExample">
    <constructor-arg type="java.lang.Boolean" value="true"/>
    <constructor-arg type="java.lang.String" value="Mayur Ramgir"/>
</bean>
</beans>
```

Constructor argument index: In case there are multiple arguments with same type for example two `int` type arguments, we can use index attribute to specify the value for the desired argument. This will resolve the ambiguity in specifying values to the arguments.

```
<beans>
<bean id="ConstructorBasedDependencyInjectionSimpleTypeExample" class="javall.fundamentals.chapter20.ConstructorBasedDependencyInjectionSimpleTypeExample">
    <constructor-arg index="0" value="true"/>
    <constructor-arg index="1" value="Mayur Ramgir"/>
</bean>
</beans>
```

Constructor argument name: You may also use name attribute to specify value to the required argument. See the following example:

```
<beans>
<bean id="ConstructorBasedDependencyInjectionSimpleTypeExample" class="javall.fundamentals.chapter20.ConstructorBasedDependencyInjectionSimpleTypeExample">
    <constructor-arg name="healthy" value="true"/>
    <constructor-arg name="name" value="Mayur Ramgir"/>
</bean>
</beans>
```

20.1.2.3.2 Setter-Based Dependency Injection

In setter-based DI, Spring container instantiates the bean by invoking a no-argument constructor or no-argument static factory method. Once the bean is instantiated, the container then uses setter methods to inject the dependencies.

Let us see the following example to understand this better.

```
package javall.fundamentals.chapter20;
public class SetterBasedDependencyInjectionExample {
    private Camera camera;
    public SetterBasedDependencyInjectionExample() {}
    public Camera getCamera() {
        return camera;
    }
    public void setCamera(Camera camera) {
        this.camera = camera;
    }
}
package javall.fundamentals.chapter20;
public class Camera {
    public Camera() {

    }
}
```

Following is the XML configuration specify the Camera dependency:

```
<bean id = "setterBasedDependencyInjectionExample" class = "javall.fundamentals.chapter20.SetterBasedDependencyInjectionExample">
    <property name = "camera" ref = "camera"/>
</bean>
<bean id = "camera" class = "javall.fundamentals.chapter20.Camera"></bean>
```

As you can see, we need to use <property> to set the dependent bean reference. Please note that the “name” attribute must match with the field name and “ref” attribute must match the “id” attribute of the dependent bean reference.



Differentiate between setter injection and constructor injection.

20.1.2.3 Field-Based Dependency Injection

In field-based DI, the Spring container injects the dependencies on fields which are annotated as @Autowired once the class is instantiated. See the following example:

```
package java11.fundamentals.chapter20;
public class FieldBasedDependencyInjectionExample {

    @Autowired
    private Camera camera;
    public FieldBasedDependencyInjectionExample() { }
}
```

In the above example, Spring will inject the *Camera* bean since it is annotated with **@Autowired**.

This looks the simplest of all, but it is highly discouraged to use because of several drawbacks, which are as follows:

1. It is expensive to use field-based DI in place of constructor or setter as Spring uses reflection to inject the annotated dependencies.
2. It does not support final or immutable declared fields, as they will get instantiated at the time of class instantiation. Only constructor-based DI supports immutable dependencies.
3. Possibility of violating the single responsibility principle as in field-based DI for a class. It is not difficult at all to end up having a lot of dependencies without even realizing that the class now focuses on multiple functionalities and violates its single responsibility principle. This problem is clearly visible in case of constructor-based injection, as the large number of parameters in the constructor will be clearly visible and give the signal of violation.
4. It creates tightly coupled code, as the main reason of using this injection is to avoid defining getters and setters and/or constructor. This leads to a scenario where only Spring container can set these fields via reflection. This is because there are no getters or setters, so the dependencies will not be set outside the Spring container, making it dependent on using the Spring container to use reflection to set the dependencies on autowired fields.
5. It creates hidden dependencies, as these fields do not have public interfaces like getters and setters, nor are they exposed via constructors. Hence, the outside world is unaware of these dependencies.

20.1.2.4 Lazy Initialized Beans

This is an interesting functionality that allows initializing a bean on first request and not on the application startup. It helps in increasing application performance, as many times, you may not need all the beans loaded on the startup. This way it enhances initialization time. However, there is a danger of discovering configuration errors later on bean request. This may interrupt the running application.

Following is an example of specifying lazy initialization:

```
<bean id = "camera" class = "java11.fundamentals.chapter20.Camera" lazy-init = "true">
</bean>
```

Now that you know both IoC and DI concepts, you can see how easy it is to use the DI approach to feed the required dependencies to the bean. It greatly simplifies the application development and helps in resource management, which is crucial for performance driven applications. There is one more concept that you should know before we discuss Spring MVC. This concept is called aspect-oriented programming.

20.1.3 Aspect-Oriented Programming

Modularity is one of the important design principals to accomplish scalability and reusability. This can be achieved by subdividing the system into smaller parts keeping business logic separate from each other and focus on one particular functional area.

In the aspect-oriented programming (AOP) world, these standalone distinct parts are called *concerns*. Some of the functions which are distinct from the application's business logic like security, auditing, logging, caching, transactions, etc., may cross multiple areas of an application and are termed as *cross-cutting concerns*. AOP greatly helps along with DI, as DI focuses on decoupling objects from each other and AOP helps in decoupling cross-cutting concerns from the objects. We can also

relate AOP to *triggers* in programming languages like Java, .Net, Perl, etc. Similar to the trigger concept, Spring AOP offers *interceptors* which intercepts an application by inserting additional functionality on method execution such as before the method or after the method, without adding the required logic in the method itself.

**QUICK
CHALLENGE**

Explain the different AOP concepts and terminologies.

Following are some of the important concepts we should know about AOP.

1. **Aspect:** The standalone distinct modules are called *aspect*. For example, logging aspect, auditing aspect, etc.
2. **Join point:** As the name suggests, this is the point where aspects can be injected to perform a specific task.
3. **Advice:** It is the actual code block, which will be executed before or after the method execution.
4. **Pointcut:** It is a set of one or more join points where the actual code will get executed. It is a predicate which matches join points to execute advice. Pointcuts are specified using expressions or patterns. See the following example:

```
@Aspect
public class LoggingAspectPointcutExample {
    @PointCut("execution(* java11.fundamentals.*.*(..))")
    private void logData() {}
}
```

In the above example, we have used a few annotations. Let us understand their meanings.

1. **@Aspect:** This annotation specifies that the class contains advice methods.
2. **@PointCut:** This annotation specifies the JoinPoint where an advice should be executed.
3. **execution(* java11.fundamentals.*.*(..)):** This specifies on which methods the given advice should be applied.
4. **Introduction:** This enables the inclusion of new methods, fields or attributes to the existing classes. It is also known as inter-type declarations in *AspectJ*. In other words, with introduction, an aspect can assert advised objects to implement a specific interface. It can also provide implementation of this interface.
5. **Target object:** It specifies the object that is being advised by one or more aspects. This object will always be a *proxied* object as Spring AOP is implemented using runtime proxies.
6. **AOP proxy:** It denotes the object to implement the aspect contracts.
7. **Weaving:** It is a process to create an advised object by linking one or more aspects with other objects or application types. There are various ways it can be done such as compile, load, or at runtime. Spring AOP performs weaving at runtime.

Types of Advice

There are total of five advices that can be used in the applications. Each advice has a specific purpose to accomplish. Following are descriptions of these advices:

1. **before:** This advice type makes sure to run the stated advice before the method execution on which it is specified.
2. **after:** This advice type makes sure to run the stated advice after the method execution on which it is specified irrespective of this method's outcome.
3. **after-returning:** This advice type makes sure to run the stated advice only after the successful completion of the method on which it is specified.
4. **after-throwing:** This advice type makes sure to run the stated advice only on method exits by throwing an exception on which it is specified.
5. **around:** This advice type makes sure to run the stated advice before and after the method execution on which it is specified. This type is more useful on the auditing and logging type of aspects.

**QUICK
CHALLENGE**

Explain Spring Aspect Oriented Programming (AOP) capabilities and goals.



20.2 | Spring Architecture

Now you have learned about IoC and DI, you must have comprehended that the Spring is built as a modular framework. Hence, you have an option to pick and choose the modules you need as per your requirement. Let us explore the modules offered by Spring Framework.

Overall, there are 20 modules that provide various functionalities. See Figure 20.2, which shows the Spring Framework's architecture diagram, to understand this better.

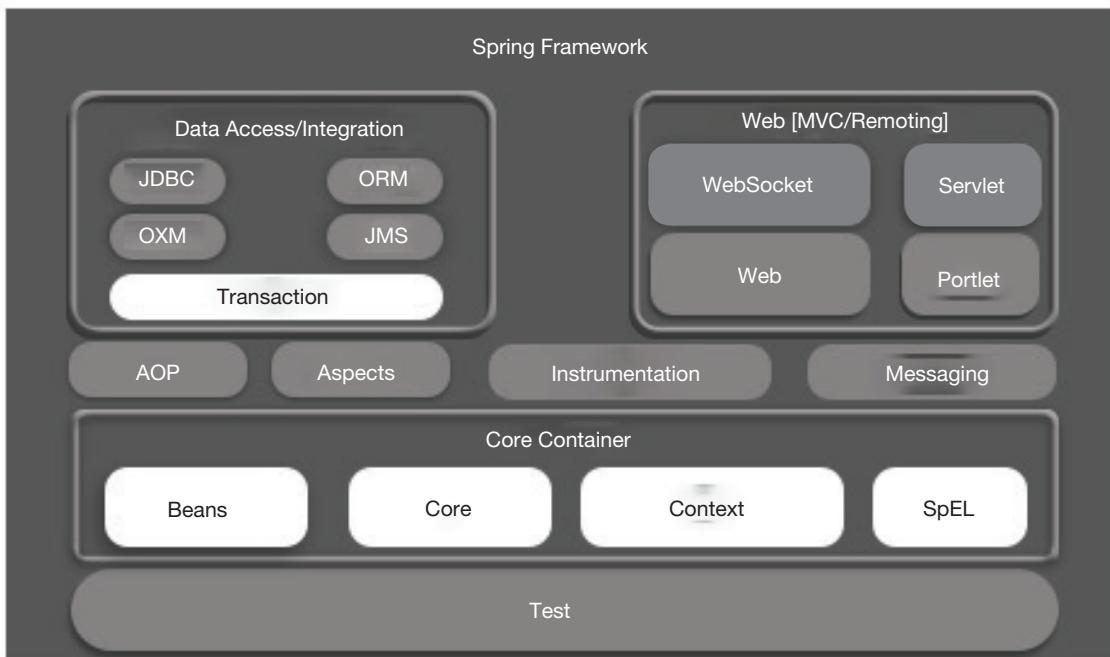


Figure 20.2 Spring framework architecture diagram.

20.2.1 Core Container

As you can see, the core container acts as the base of Spring Framework, which contains modules like Beans, Core, Context, and Expression Language (SpEL). All other modules rest on top of the core container. Let us study these modules in detail.

- Core module:** This is responsible for providing fundamental features like *IoC* and *DI*, which are the crucial parts of the framework that provide modularity.
- Bean module:** This provides implementation of a factory pattern known as *BeanFactory*.
- Context module:** This module is the provider of a well-known interface called *ApplicationContext*. The Context module uses the base offered by Core and Bean modules. *ApplicationContext* is the interface that provides configuration information to the application, which is necessary to instantiates the required beans. Spring builds up this interface upon starting the application and it is read-only at run-time but it can be reloaded if required. Many classes implement this interface to provide various configuration options. *ApplicationContext* offers many bean factory methods, which can be used to access different application components. It provides various functionalities such as allowing publishing events to register listeners, loading file resources, supporting internationalization by resolving messages, etc.
- SpEL module:** This module provides a prevailing tool to query and manipulate an object graph at runtime. It supports various functionalities such as bean references, calling constructors, method invocation, collection projection, collection selection, relational operators, regular expressions, Boolean and relational operators, assignment, accessing properties, array construction, accessing arrays, accessing lists, accessing maps, literal expression, class expressions, user defined functions, inline maps, inline lists, variables, templated expressions, and ternary operator. It also offers various interfaces and classes such as ExpressionParser interface, EvaluationContext interface, Expression interface, SpelExpression class, SpelExpressionParser class, and StandardEvaluationContext.



Does Spring provide transaction management?

20.2.2 Data Access/Integration

This section looks after accessing, storing, and manipulating data. This layer contains modules such as ORM, OXM, JDBC, JMS, and Transaction. Following are the descriptions of each module.

1. **ORM module:** ORM stands for object-relationship mapping, which offers an easy way to persist objects without writing database queries. It binds objects to the database tables, which hides the details of SQL queries from application logic. With ORM, objects can deal with other objects without worrying about underlying persistence mechanism. Spring does not offer ORM implementation but it provides a mechanism to integrate with external ORM tools like Hibernate, JPA, JDO, iBatis, etc. In the Chapter 21, we will explore Hibernate in detail.
2. **JDBC module:** This module provides an abstraction layer for JDBC, which is a Java API to interact with database to perform CRUD operations (CRUD stands for create, read, update, and delete). It uses JDBC drivers to connect with different providers databases.
3. **OXM module:** OXM stands for Object XML Mapping, which offers a mechanism to convert an object to an XML document and vice versa. XML is one of the important formats of transferring data to remote applications. With OXM, it is easier to convert the transmitted data into an object and prepare objects to transfer over network. The conversion from object to XML known as XML Marshalling or XML Serialization, and from XML to object known as Unmarshalling or Deserialization. This module is an extendible one and can integrate with other frameworks such as Castor, JAXB, and XStream.
4. **JMS module:** In distributed and modular systems, messaging plays a crucial role for communication between loosely coupled elements. Java provides an API called Java Message Service (JMS), which offers a mechanism to create, send, and read messages in a reliable and asynchronous way.
5. **Transaction management module:** Transaction management is a critical feature in RDBMS to promise data integrity and consistency. Database transactions are a series of actions that are considered as a single unit of work, which must complete totally or not at all. This module offers a reliable abstraction for transaction management which not only supports declarative transaction management but also offers a programmatic transaction management simpler than Java Transaction API (JTA). This module is consistent across different APIs such as JTA, Hibernate, Java Persistent API (JPA), JDBC, and Java Data Objects (JDO).

QUICK CHALLENGE

Explain the benefits of ORM.

20.2.3 Web Container

This section provides various frameworks that are useful for web related applications, which offers integration with other popular MVC frameworks such as Struts and JSF. It has various modules such as Web, Web-Servlet, Web-Sockets, and Web-Portlet. Let us explore in detail.

1. **Web module:** This module offers web-related functionalities such as multiple file uploads, integration to IoC using a servlet, and web application context.
2. **Web-servlet module:** This module is an implementation of Model–View–Controller (MVC) pattern. It offers a clean separation between domain model code and web interface.
3. **Web-socket module:** This module provides support of Web-Socket, which is a bidirectional communication between server and client.
4. **Web-portlet module:** This module provides similar functionality as Web-WebSocket module for implementing MVC in a portlet environment.



What is the role of web container?

20.2.4 Other Modules

1. **AOP module:** This module provides Aspect Oriented Programming support to Spring Framework. As we have learned in Section 21.1.3, AOP provides a simple mechanism to implement cross-cutting concerns.
2. **Aspects module:** This module supports integration with AspectJ.
3. **Testing module:** This module provides provision to implement integration and unit tests. With the help of this module, JUnit or TestNG type of frameworks can be used in Spring Framework.
4. **Instrumentation module:** This module provides class instrument support and classloader implementations, which are useful in various application servers.
5. **Messaging module:** This module provides foundation for messaging-based applications, which contains a set of annotations for mapping messages to methods.

Due to the modularity of the Spring Framework, you are free to choose the modules you need without breaking any core framework's functionality.

20.3 | Spring MVC



Spring MVC is one of the most popular Java website frameworks to program website applications. Similar to other industry technologies, it makes use of the model view design. All the general Spring features such as DI and IoC are implemented by Spring MVC.

To use the Spring framework, Spring MVC makes use of class DispatcherServlet. This class processes an incoming request and assigns it with models, controller, views, or any of the right resource.

If you are unfamiliar with MVC pattern, then bear in mind that it is a software architectural pattern. The data of the application is referred to as the model which can be a single or multiple objects. For the business logic, a controller is used which acts as a supervisor. The end user is provided a perspective through a "view".

In Spring MVC, any class which uses the annotation "@Controller" is the controller of the application. For views, a combination of JSP and JSTL can be utilized, though developers can also use other technologies. For the front controller, the DispatcherServlet class is used in Spring MVC.

When a request arrives, it is discovered by the DispatcherServlet class. This class gets the required information from the XML files and delivers the request to the controller. Subsequently, the controller generates a ModelAndView object. Afterwards, the DispatcherServlet class processes the view resolver and calls the appropriate view.

20.3.1 DispatcherServlet

The DispatcherServlet is used for request processing. It has to be mapped and declared in accordance with the Servlet specification. This servlet utilizes configurations in Spring to get information related to the delegate components for exception handling, view resolution, and request mapping.

20.3.1.1 Context Hierarchy

For configuration purposes, the DispatcherServlet requires the WebApplicationContext. It is linked with the Servlet and ServletContext. It is also attached to the ServletContext so the methods which are associated with it can utilize the RequestContextUtils.

For most of the applications, one WebApplicationContext is enough. However, sometimes there is a context hierarchy in which a single root WebApplicationContext is used for sharing between more than one instance of DispatcherServlet. Each of them has its configuration of own child WebApplicationContext.

Generally, the root WebApplicationContext stores data repositories, business services, and other infrastructure beans. These are shared between the Servlet instances. It is possible to override these beans as they are inherited via the WebApplicationContext of the Servlet children.

Special Bean Types

1. **HandlerMapping:** It is used with a handler for mapping requests. It uses the HandlerMapping implementation.
2. **HandlerAdapter:** It uses the DispatcherServlet for invoking handlers which are mapped with a request.

3. **HandlerException resolver:** It is used for resolving exceptions where they are mapped with handlers and error views of the HTML.
4. **View resolver:** It is used to resolve the view names which rely on logical Strings.
5. **Locale resolver:** It is used for resolving a client's locale or timezone.
6. **Themes resolver:** It is used to resolve the themes of a web application.
7. **MultipartResolver:** It is used for abstraction in order to help with parsing a request which has multiple parts.
8. **FlashMapManager:** It is used for storing and retrieving the input/output FlashMap.

20.3.2 Spring MVC Processing

The DispatcherServlet is used for the following:

1. It searches the WebApplicationContext and attaches the request in the form of an attribute so other components such as controller can use it.
2. The request is bound to the locale resolver which allows the process' elements in resolving the locale while processing and preparing data, rendering the view, or other requests.
3. Requests are bound to the theme resolver which allows views and other elements to use the theme.
4. When a multipart file resolver is defined, all the parts of the request are processed. After multipart are discovered, the MultipartHttpServletRequest is wrapped to help with processing.
5. When the right handler is found, controllers, preprocessors, postprocessors, and others in the handler's executive chain are executed so the model is prepared or rendered. For annotated controllers, it is possible to render the response.
6. The view is rendered when a model is returned. When it is not returned, there is no need for view rendering as the request is already processed.
7. The declared beans in HandlerExceptionResolver resolve exceptions which come up in the request processing.

The Spring DispatcherServlet helps in returning the “last-modification-date”, which is defined in the Servlet API. In order to assess this date, a simple processing is used.

1. The DispatcherServlet searches for the right handler mapping.
2. The DispatcherServlet then checks if the handler has implemented the “LastModified” interface.
3. If verified, the “long getLastModified” method value of the interface is sent to the client.



Can you configure Spring application with Configuration file and Annotations?

20.4 | Interception



In all of the Spring MVC HandlerMapping implementations, interceptors are supported. They are used to implement certain functionality for specific requests. For instance, they can be used to check for a principal. It is necessary for an interceptor to implement “HandlerInterceptor”, which is included in the org.springframework.web.servlet having the following methods for preprocessing and post-processing.

1. **preHandle():** It is used for the time period before the handler is executed.
2. **postHandle():** It is used when the handler has been executed.
3. **afterCompletion():** It is used after the request has been completed the execution.

20.5 | Chain of Resolvers



When you declare multiple beans of HandlerExceptionResolver, you can create an exception resolver chain in your SpringMVC application where you have to configure and define the order properties accordingly. If the order property is high, then the exception resolver is processed accordingly. With respect to the HandlerExceptionResolver, it returns the following.

1. An empty ModelAndView in case the resolver handled the exception.
2. A Model and View which refers to an error view.
3. In case the exception is not resolved, it returns “null” for other resolvers. On the other hand, if the exception is still staying till the end, the Servlet container can be used to bubble it up.



Can you manage concurrent sessions using Spring MVC? If so, what are the things you need to consider?

20.6 | View Resolution



The View and ViewResolver interfaces are used by the SpringMVC, which helps in rendering models present in a browser while avoiding dependence on a certain technology for viewing. The ViewResolver is used to map actual views and view names. The View manages the preparation data before it is passed over to a certain technology for viewing.

It is possible to use multiple resolver beans to chain the view resolvers. Sometimes, the order property is used for defining ordering with it. To configure the view resolution, you just have to add the ViewResolver beans in the configuration of your SpringMVC application.

A Basic Example

Consider the following example for a basic Spring Web MVC project. Use Maven and add dependency in the pom.xml file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.introspring</groupId>
<artifactId>SpringMVC</artifactId>
<packaging>war</packaging>
<version>0.0.1-SNAPSHOT</version>
<name>SpringMVC Example</name>
<url>http://maven.apache.org</url>
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>5.1.1.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>servlet-api</artifactId>
<version>3.0-alpha-1</version>
</dependency>
</dependencies>
<build>
<finalName>SpringMVC</finalName>
</build>
</project>
```

Now generate a class for the controller and add the following two annotations. The @Requestmapping annotation is required for mapping the appropriate URL name with the class. The @Controller class is simply used to treat a class as the controller.

```

package com.introspring;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ContClass {
@RequestMapping("/")
    public String show()
    {
        return "index";
    }
}

```

In the XML file, make sure to write about the DispatcherServlet class which is serves as the front controller.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>Model View Controller</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>

```

In the spring-servlet.xml file, define the components for the views. This XML file must be placed in the WEB-INF directory.

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!-- Specify the functionality to scan components -->
    <context:component-scan base-package="com.introspring" />

    <!--Specify the functionality for formatting, validation, and conversion -->
    <mvc:annotation-driven/>
</beans>

```

Generate an index JSP page and write the following.

```
<html>
<body>
<p>We are learning Spring MVC</p>
</body>
</html>
```

As an output, you can see “We are learning Spring MVC” text on your page. This message was delivered to you through the logic of the Controller.

20.7 | Multiple View Pages

In this example, we would apply redirection between two view pages. Begin by adding dependencies in the pom.xml file.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
```

Afterward, you have to generate a page for the request. To do this, create a JSP page and add a hyperlink in it.

```
<html>
<body>
<a href="Success">Hit This Link...</a>
</body>
</html>
```

Now generate a class for the controller which can process the JSP pages and return them. For mapping of the class, use the @Requestmapping annotation. We would add our “href” value in the following code so our Controller can easily manage and view our HTML elements.

```
package com.ith;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ContClass {
    @RequestMapping("/success")
    public String reassign()
    {
        return "view";
    }
    @RequestMapping("/success again")
    public String show()
    {
        return "final";
    }
}
```

In the web.xml file, place a controller entry.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Now, define the bean for XML file. You have to add the view resolver in the component of the view. For the ViewResolver, the InternalResourceViewResolver class can be used. The XML file must be placed in the WEB-INF directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!--Add functionality to scan components -->
    <context:component-scan base-package="com.ith" />

    <!--Add functionality for formatting, validation, and conversion -->
    <mvc:annotation-driven/>
    <!--Specify the view resolver for the Spring MVC -->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceView-
    wResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

Now generate the components for the view. In the view.jsp, write the following.

```
<html>
<body>
<a href="successagain">Spring Tutorial</a>
</body>
</html>
```

In the last.jsp, write the following.

```
<html>
<body>
<p>Hello User, Let's Learn Spring MVC to Master the Java Backend</p>
</body>
</html>
```

When you run this application, you would first get a web page with a hyperlink. When you will click on it, it sends a request which is managed by the controller to reply with a response in the form of another page. Hit the link on the new page and you would be again redirected to another page, which would show you a welcome message. In this way, we have successfully gone over multiple web pages in Spring MVC.

20.8 | Multiple Controllers

Till now, we have been using a single controller in our examples. However, it is possible to generate multiple controllers in Spring MVC at the same time. Each of the controller class must be mapped properly with the @Controller annotation.

Begin by adding the dependencies in the pom.xml file as we have done before.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
```

Then generate your initial page which would be initially viewed by the user. We would add two different hyperlinks for each of our controllers.

```
<html>
<body>
<a href="hiuser1">First User</a> ||
<a href="hiuser2">Second User</a>
</body>
</html>
```

Generate two classes for controller which can process and return the specified view page. For the first class, write the following code.

```
package com.ith;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class User1 {
    @RequestMapping("/hiuser1")
    public String show()
    {
        return "vp1";
    }
}
```

For the second class, write a similar piece of code.

```
package com.ith;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class User2 {
    @RequestMapping("/hiuser2")
    public String show()
    {
        return "vp2";
    }
}
```

Place an entry for the controller in the web.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Then specify the bean in the XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- Add functionality to scan component-->
    <context:component-scan base-package="com.ith" />
    <!-- Add functionality for validation, formatting, and conversion -->
    <mvc:annotation-driven/>
    <!--Specify the view resolver in Spring MVC -->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

Generate the components for the view. In the first vp1.jsp, write the following.

```
<html>
<body>
<p> In this example we have used two controllers.</p>
</body>
</html>
```

In the second vp2.jsp page, write the following code.

```
<html>
<body>
<p> In this example we have used two view pages.</p>
</body>
</html>
```

Run this code. In the output, the user would be presented two links. When the user clicks on the first page, they would be redirected to a new page and displayed a text. Clicking on the second link will take the user to a different page with different text. This is exactly how it happens in real world browsing. Hence, whenever you are clicking on links on the websites, then behind the scenes, each link is configured to redirect to a different page by their respective controllers.



Can you add security on Spring controllers to avoid denial of service attack? Explain.



20.9 | Model Interface

In the Spring MVC ecosystem, the model serves as a container which can be used to store the application's data. This data can be anything such as a record from the DB, an object, or simply a String. The controller must have the Model interface. The HttpServletRequest object processes the data given by a user and sends it to this interface. Any view page can then access that piece of data from model's interface.

As an example, let us generate a login page which has a username and password, a common use case in the web world. For validation, we can define a certain value. Add the required dependencies in the pom.xml.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
```

Now generate the login page which would get the user's credentials. We will name it as index.jsp.

```

<html>
<body>
<form action="hiuser">
UserName : <input type="text" name="name"/> <br><br>
Password : <input type="text" name="pass"/> <br><br>
<input type="submit" name="submit">
</form>
</body>
</html>

```

Now, generate the controller class in which the HttpServletRequest would be responsible to intercept the data which is typed by the user on the HTML form. The Model interface saves the data for the request and sends it to the view page.

```

import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ContClass{
    @RequestMapping("/hiuser")
    public String show(HttpServletRequest r,Model md)
    {
        //read the data from the form
        String name=r.getParameter("name");
        String pass=r.getParameter("pass");
        if(pass.equals("abc12"))
        {
            String message="Welcome "+ name;
            //add a message to the model
            md.addAttribute("message", message);
            return "vp";
        }
        else
        {
            String msg="We apologize Mr. "+ name+". This is the wrong password";
            md.addAttribute("message", message);
            return "ep";
        }
    }
}

```

Now, open the web.xml file and add the controller's entry.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>

```

Afterward, configure the bean by setting up the spring-servlet.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- Add functionality to scan component-->
    <context:component-scan base-package="com.ith" />
    <!-- Add functionality for validation, formatting, and conversion -->
    <mvc:annotation-driven/>
    <!--Specify the view resolver in Spring MVC -->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

For the components of the view, add two .jsp pages in the WEB-INF/jsp directory. The first page vp.jsp should look like the following. See how we are using the backend code to display our code dynamically on user's view by adding the "\${message}" parameter.

```
<html>
<body>
${message}
</body>
</html>
```

The second page ep.jsp should show something like the following.

```
<html>
<body>
${message}
<br><br>
<jsp:include page="/index.jsp"></jsp:include>
</body>
</html>
```

When the user gets the username page, then there are two possible scenarios.

1. If the user types the correct password "abc12" and hits the submit button, then the controller would take them into a new page with a greeting.
2. On the other hand, a wrong password means that they would remain in the same page with a warning.

The basic idea behind the "login" functionality you see in websites is pretty much the same.

20.10 | RequestParam

The @RequestParam annotation reads data from the form and applies automatic binding for the method's available parameters. It does not consider the HttpServletRequest object for reading data. The annotation also applies mapping for the request parameter. If the type of the parameter in the method is "Map" along with the definition for the name of the

request parameter, then that parameter is changed into a Map. Otherwise, the name and values for the request parameter are placed in the map parameter.

As an example, let us generate a login page. Begin by placing the required dependencies in the pom.xml.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
```

Now, generate the page for the request which would take the credentials of the user. This page is saved as index.jsp.

```
<html>
<body>
<form action="hiuser">
Student Name : <input type="text" name="name"/> <br><br>
Password : <input type="text" name="pass"/> <br><br>
<input type="submit" name="submit">
</form>
</body>
</html>
```

Now, generate a controller class. In this class, the @RequestParam annotation reads the data from the form and applies binding for the request parameter.

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class ContClass {
    @RequestMapping("/hiuser")
    //read the provided form data
    public String display(@RequestParam("name") String name, @RequestParam("pass") String pass, Model md)
    {
        if(pass.equals("abc12"))
        {
            String message="Welcome "+ name;
            //write a message for the model
            md.addAttribute("message", message);
            return "vp";
        }
        else
        {
            String msg="We apologize "+ name+". The typed password is wrong";
            md.addAttribute("message", message);
            return "ep";
        }
    }
}
```

For the vp page, write the following.

```
<html>
<body>
${message}
</body>
</html>
```

For the ep page, write the following.

```
<html>
<body>
${message}
<br><br>
<jsp:include page="/index.jsp"></jsp:include>
</body>
</html>
</html>
```

The output is similar to the previous example. However, the controller's working changed. In this example, we did not use the HttpServletRequest class. Instead, we used our @RequestParam annotation for our HTML elements. Through the annotation, we were able to apply automatic binding on HTML elements.

20.11 | Form Tag Library

Form tags in Spring MVC are the basic components of web pages. These tags can be reused and reconfigured according to the requirements of the users. They are extremely helpful for JSP for the readability and maintainability factors.

The library for the form tags is in the spring-webmvc.jar. For turning on the library's support, some configuration is required. Place the following code in the start of the JSP web page.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

The container tag is a sort of parent tag which stores all of the library's tag. Such tag creates a form tag for HTML.

```
<form:form action="nextFormPath" modelAttribute=?abc?>
```

Following are one of the most common form tags in Spring MVC.

1. **form:form** – Used to store all of the other tags.
2. **form:input** – Creates the text field for the user input.
3. **form:radiobutton** – Creates the radio button which can be marked or unmarked by the user.
4. **form:checkbox** – Creates the checkboxes which can be checked or unchecked by the user.
5. **form:password** – Creates a field for user to type password.
6. **form:select** – Creates a drop-down list for the user to choose an option.
7. **form:textarea** – Creates a field for text which spans multiple lines.
8. **form:hidden** – Creates an input field which is hidden.

20.12 | Form Text Field

The form text field creates an HTML input tag through the use of the bound value. For instance, consider a form where users can type their details to make a reservation in a restaurant. This example mirrors the real world reservations and booking structure.

To begin with, add the required dependencies in the pom.xml file.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
```

Generate a bean class now. This class can store variables and the getter setter methods for the input fields of the form.

```
public class Reservation {
    private String fName;
    private String lName;
    public Reservation() { }
    public String getFname() {
        return Fname;
    }
    public void setFname(String Fname) {
        this.Fname = Fname;
    }
    public String getLname() {
        return Lname;
    }
    public void setLname(String Lname) {
        this.Lname = Lname;
    }
}
```

Now add a Controller class.

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@RequestMapping("/reservation")
@Controller
public class ContClass {
    @RequestMapping("/bkfrm")
    public String bkfrm(Model md)
    {
        //generate an object for reservation
        Reservation rsv=new Reservation();
        //pass the object to the model
        md.addAttribute("reservation", rsv);
        return "rsv-page";
    }
    @RequestMapping("/subfrm")
    // @ModelAttribute applies binding for the data of the form to the object
    public String submitForm(@ModelAttribute("reservation") Reservation rsv)
    {
        return "confirmation-form";
    }
}
```

You can then add the controller's entry in the web.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

In the spring-servlet.xml file, place the following code.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!--Enables the scanning of components -->
    <context:component-scan base-package="com.ith" />
    <!--Enables formatting, validation, and conversion -->
    <mvc:annotation-driven/>
    <!-- Define Spring MVC view resolver -->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalRe-
sourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

To generate the requested page, place the following code in the index.jsp file.

```
<!DOCTYPE html>
<html>
<head>
    <title>Reservation Form for the Restaurant</title>
</head>
<body>
<a href="reservation/bkfrm">Click this link to get a booking at the restaurant</a>
</body>
</html>
```

For the reservation page, place the following code.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
    <title>Restaurant Form</title>
</head>
<h3>Reservation Form for Restaurant</h3>
<body>
    <form:form action="subfrm" modelAttribute="reservation">
        First name: <form:input path="Fname" />
        <br><br>
        Last name: <form:input path="Lname" />
        <br><br>
        <input type="submit" value="Submit" />
    </form:form>
</body>
</html>
```

In the confirmation page, place the following code.

```
<!DOCTYPE html>
<html>
<body>
<p>Your reservation is confirmed successfully. Please, re-check the details.</p>
First Name : ${reservation.Fname} <br>
Last Name : ${reservation.Lname}
</body>
</html>
```

In the output, a hyperlink will take a user to a reservation form for the restaurant. When the user completes typing the details and submits it, then they are displayed their information as part of the rechecking procedure.

20.13 | CRUD Example

If you are learning a web framework, then it is important to equip yourself with enough knowledge that you can make a basic CRUD application. Such functionality is one of the most common client requirements and therefore getting the hang of it can be extremely useful for your career. To begin with, generate a new table in your database titled “students”. Configure and enter the following fields in it.

1. ID
2. Name
3. GPA
4. Department

Now come to Eclipse IDE and begin the management of your dependencies by adding the following in the pom.xml file.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.11</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
```

Subsequently, you have to create your bean class “Student” which contains variables according to the columns of your database.

```
public class Student {
    private int id;
    private String name;
    private float GPA;
    private String department;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public float getGPA() {
        return GPA;
    }
    public void setGPA(float GPA) {
        this.GPA = GPA;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
}
```

For the controller class, write the following code.

```
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.ith.beans.Student;
import com.ith.dao.StudentDao;
@Controller
```

```

public class StudentController {
    @Autowired
    StudentDao sDao;//Use the XML file for dao injection
    /* To take input, it generates a form. You can also see the reserved request attribute
    in the form of "command". This attribute shows the data pertaining to the object in the
    form.
    */
    @RequestMapping("/studentform")
    public String displayform(Model md){
        md.addAttribute("command", new Student());
        return "studentform";
    }
    /* This is done in order to use the database for storing data. The model
    object receives request data from the @ModelAttribute. It is also necessary to add
    RequestMethod.Post method as by default, the request type is set to GET.
    */
    @RequestMapping(value="/store",method = RequestMethod.POST)
    public String store(@ModelAttribute("student") Student student){
        sDao.store(student);
        return "redirect:/viewstudent";//
    }
    /* To show the student list which are stored in the model object */
    @RequestMapping("/viewstudent")
    public String viewstudent(Model md){
        List<Student> list=sDao.getStudents();
        md.addAttribute("list",list);
        return "viewstudent";
    }
    /* It shows the data of object from the form in accordance with the provided id.
    To add data in the variable, we have used the @PathVariable. */
    @RequestMapping(value="/editstudent/{id}")
    public String edit(@PathVariable int id, Model md){
        Student student=sDao.getStudentById(id);
        md.addAttribute("command",student);
        return "studentupdateform";
    }
    /* It edits the object in the model. */
    @RequestMapping(value="/editsave",method = RequestMethod.POST)
    public String editsave(@ModelAttribute("student") Student student){
        sDao.update(student);
        return "redirect:/viewstudent";
    }
    /* It is used for the deletion of a record. */
    @RequestMapping(value="/deletestudent/{id}",method = RequestMethod.GET)
    public String delete(@PathVariable int id){
        sDao.delete(id);
        return "redirect:/viewstudent";
    }
}

```

Create a DAO class like the following.

```

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import com.ith.beans.Student;

public class StudentDao {
    JdbcTemplate tpe;

    public void setTemplate(JdbcTemplate tpe) {
        this.tpe = tpe;
    }
    public int save(Student s){
        String sql="insert into students (name,GPA,department) values ('"+s.
getName()+"','"+s.getGPA()+"','"+s.getDepartment()+"')";
        return tpe.update(sql);
    }
    public int update(Student s){
        String sql="update students set name='"+s.getName()+"', GPA='"+s.
getGPA()+"',department='"+s.getDepartment()+"' where id='"+s.getId()+"';
        return tpe.update(sql);
    }
    public int delete(int id){
        String sql="delete from students where id='"+id+"'";
        return tpe.update(sql);
    }
    public Student getStudentById(int id){
        String sql="select * from students where id=?";
        return tpe.queryForObject(sql, new Object[]
{id},new BeanPropertyRowMapper<Student>(Student.class));
    }
    public List<Student> getStudents(){
        return tpe.query("select * from students",new RowMapper<Student>(){
            public Student mapRow(ResultSet rs, int row) throws SQLException {
                Student s=new Student();
                s.setId(rs.getInt(1));
                s.setName(rs.getString(2));
                s.setGPA(rs.getFloat(3));
                s.setDepartment(rs.getString(4));
                return s;
            }
        });
    }
}

```

In the web.xml file, add the controller.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.
com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.
sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>

```

In the XML file, specify the bean.

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd">
<context:component-scan base-package="com.ith.controllers"></context:component-scan>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/jsp/"></property>
<property name="suffix" value=".jsp"></property>
</bean>

<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
<property name="url" value="jdbc:mysql://localhost:3306/test"></property>
<property name="username" value=""></property>
<property name="password" value=""></property>
</bean>

<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="ds"></property>
</bean>

<bean id="sDao" class="com.ith.dao.StudentDao">
<property name="template" ref="jt"></property>
</bean>
</beans>
Generate a requested HTML page.
<a href="studentform">Add Students</a>
<a href="viewstudent">View Students</a>
In the JSP student form, add the following.
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<h1>Add New Student</h1>
<form:form method="post" action="save">
<table >
<tr>
<td> StudentName : </td>
<td><form:input path="name" /></td>
</tr>
<tr>
<td>GPA :</td>
<td><form:input path="GPA" /></td>
</tr>
<tr>
<td>Department :</td>
<td><form:input path="department" /></td>
</tr>
<tr>
<td> </td>
<td><input type="submit" value="Save" /></td>
</tr>
</table>
</form:form>

```

By changing the project name in the following file, add the following code in the studenteditform.jsp file.

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

    <h1>Edit Student</h1>
    <form:form method="POST" action="/SpringCRUDExample/editsave">
        <table >
            <tr>
                <td></td>
                <td><form:hidden path="id" /></td>
            </tr>
            <tr>
                <td>Name : </td>
                <td><form:input path="name" /></td>
            </tr>
            <tr>
                <td>GPA :</td>
                <td><form:input path="GPA" /></td>
            </tr>
            <tr>
                <td>Department :</td>
                <td><form:input path="department" /></td>
            </tr>
            <tr>
                <td> </td>
                <td><input type="submit" value="Edit Save" /></td>
            </tr>
        </table>
    </form:form>
```

In the viewstudent file, add the following code.

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

    <h1>Student List</h1>
    <table border="2" width="70%" cellpadding="2">
        <tr><th>Id</th><th>StudentName</th><th>GPA</th><th>Department</th><th>Edit</th><th>Delete</th></tr>
        <c:forEach var="student" items="${list}">
            <tr>
                <td>${student.id}</td>
                <td>${student.name}</td>
                <td>${student.GPA}</td>
                <td>${student.department}</td>
                <td><a href="editstudent/${student.id}">Edit</a></td>
                <td><a href="deletestudent/${student.id}">Delete</a></td>
            </tr>
        </c:forEach>
    </table>
    <br/>
    <a href="studentform">Add New Student</a>
```

When this application is run, the user gets two links; they can either add a student or see the student table. Since this is the first time you are using this application, first focus on adding the entries. Click the add button, and then you are provided with

a form to type the required details. When these details are completed, then it can be saved and the data would then be stored in the database. A user can then use this step multiple times to populate the table. After generating a table, you can modify or delete the data in the table. Click on either one of them and perform your required task. If you choose “Edit” then you can revisit the form and modify any detail. On the other hand, if you click on “Delete” then you can permanently remove the record from your database. The database would update accordingly.

You can use the logic implemented in this application in various other applications. For instance, instead of creating a student list, you can modify the details and make it a grocery shopping list where you can always adjust your items.

20.14 | File Upload in Spring MVC

In Spring MVC, you can easily incorporate the functionality to upload files in various formats. Consider the following example.

At the start, you have to download and load the Spring Core, Spring Web, commons-fileupload.jar, and commons-io.jar file.

Afterward, place the commons-io and fileupload.jar files. In the spring-servlet.xml file, write the following:

```
<bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>
Generate a form for the submission of file.
<form action="savefile" method="post" enctype="multipart/form-data">
Choose a File: <input type="file" name="file"/>
<input type="submit" value="Upload the File"/>
</form>
```

For the controller,

```
@RequestMapping(value="/savefile",method=RequestMethod.POST)
public ModelAndView upload(@RequestParam CommonsMultipartFile file,HttpSession session){
    String pth=session.getServletContext().getRealPath("/");
    String fname=file.getOriginalFilename();
    System.out.println(pth+" "+fname);
    try{
        byte b[]={};
        BufferedOutputStream bos=new BufferedOutputStream(
            new FileOutputStream(path+"/"+fname));
        bos.write(b);
        bos.flush();
        bos.close();
    }catch(Exception e){System.out.println(e);}
    return new ModelAndView("upload-success","fname",pth+"/"+fname);
}
```

To show your image in JSP, add the following code.

```
<h1>The Upload Is Successful.</h1>

To store the files in your project, generate a directory, "images".
<a href="uploadform">Upload the Image</a>
```

In the Student class, write the following:

```

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.commons.CommonsMultipartFile;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class StudentController {
    private static final String UPLOAD_DIRECTORY = "/images";

    @RequestMapping("uploadform")
    public ModelAndView uploadForm() {
        return new ModelAndView("uploadform");
    }

    @RequestMapping(value="savefile",method=RequestMethod.POST)
    public ModelAndView saveimage( @RequestParam CommonsMultipartFile file,
        HttpSession session) throws Exception{
        ServletContext context = session.getServletContext();
        String pth = context.getRealPath(UPLOAD_DIRECTORY);
        String fname = file.getOriginalFilename();
        System.out.println(pth+" "+fname);
        byte[] b = file.getBytes();
        BufferedOutputStream str =new BufferedOutputStream(new FileOutputStream(
            new File(pth + File.separator + fname)));
        str.write(b);
        str.flush();
        str.close();
        return new ModelAndView("uploadform","filesuccess","The file is saved successfully!");
    }
}

```

In the web.xml file, write the following code:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>

```

Subsequently, build a bean and type the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="com.ith"></context:component-scan>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<property name="prefix" value="/WEB-INF/jsp/"></property>
<property name="suffix" value=".jsp"></property>
</bean>

<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>

</beans>
```

The action for the form must be “post” in the following format.

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<!DOCTYPE html>
<html>
<head>
<title>Upload the Image</title>
</head>
<body>
<h1>File Upload Example in Spring MVC</h1>

<h3 style="color:blue">${filesuccess}</h3>
<form:form method="post" action="savefile" enctype="multipart/form-data">
<p><label for="image">Pick any image</label></p>
<p><input name="file" id="fileToUpload" type="file" /></p>
<p><input type="submit" value="Upload"></p>
</form:form>
</body>
</html>
```

In the output, the user first got a hyperlink. After clicking and redirecting into a new web page, the user was able to find a screen with a file upload option. When you upload a file or an image through this button, it would not show in the browser (or the client side). Instead, the file or image is sent to the server. Have your path printed via the console of the server and check the file.

20.15 | Validation in Spring MVC

Validation in Spring MVC is required for the restriction of user input so only the authorized users can proceed through a website.

In order to execute this validation, the Bean Validation API is used by the Spring MVC. This validation can be used for both the client-side and server-side programming. To use annotations and implement restrictions on object model, we use the Bean Validation API. By validation, we can validate a regular expression, a number, length, and other related input. It is also possible to offer some custom validations.

This API requires implementation because it is a type of specification. Hence, it makes use of the Hibernate Validator. Usually, the following annotations are a recurring theme in validation.

1. **@Min** – It calculates and ensures that a given number is equal to or greater than a provided value.
2. **@Max** – It calculates and ensures that a given number is equal to or less than a provided value.
3. **@NotNull** – It calculates and ensures that no null value is possible for a field.
4. **@Pattern** – It calculates and ensures that the provided regular expression is followed by the sequence.

Let us work on an example which contains a form with basic input fields. We would use the “*” sign to represent mandatory fields that must be filled in or else the form causes an error. Begin by adding the required dependencies in the pom.xml file.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
Now, generate a bean class like the following.
import javax.validation.constraints.Size;

public class Student {
    private String sname;
    @Size(min=1,message="required")
    private String pwd;

    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
}
```

For your controller class, write the following code where the `@Valid` annotation implements rules for validation to the given object while the interface “`BindingResult`” stores the validation’s output.

```
import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class StudentController {
    @RequestMapping("/hello")
    public String show(Model md)
    {
        md.addAttribute("stu", new Student());
        return "vp";
    }
    @RequestMapping("/helloagain")
    public String subfrm( @Valid @ModelAttribute("stu") Student s, BindingResult br)
    {
        if(br.hasErrors())
        {
            return "vp";
        }
        else
        {
            return "last";
        }
    }
}
```

In the `web.xml` file, adjust the following settings for the controller.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Now specify the bean,

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!--Add functionality to scan component -->
    <context:component-scan base-package="com.ith" />
    <!--Add functionality for formatting, conversion, and validation -->
    <mvc:annotation-driven/>
    <!--Specify the view resolver for Spring MVC -->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalRe-
sourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>
```

For the request page, generate an index.jsp file and place the following code.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<body>
<a href="hello">Hit This Link</a>
</body>
</html>
```

For the view components, generate the vp.jsp page with the following code.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<style>
.error{color:blue}
</style>
</head>
<body>
<form:form action="hiuser" modelAttribute="stu">
Student Name: <form:input path="sname"/> <br><br>
Password(*) : <form:password path="pwd"/>
<form:errors path="pass" cssClass="error"/><br><br>
<input type="submit" value="submit">
</form:form>
</body>
</html>
```

And finally, add a last.jsp page:

```
<html>
<body>
Student Name: ${stu.sname} <br><br>
Password: ${stu.pwd}
</body>
</html>
```

Run this application. The user first finds a hyperlink which redirects to a web page. Afterward, the name and password details of the user are typed. However, this time the “password” field is displayed as “required” when you do not enter it. After typing the password, the screen shows your name along with your password.

20.16 | Validation with Regular Expression

In Spring MVC, we can use validation of user input with a specific technique, which is also known as *regular expression*. In order to utilize the validation for regular expression, you have to use the @Pattern annotation. In such a strategy, the regexp attribute is used with the required expression along with our annotation.

Add dependencies in the pom.xml file.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.12</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

Now make the Student.java class which is our bean class.

```
import javax.validation.constraints.Pattern;
public class Student {
    private String sname;
    @Pattern(regexp="^[a-zA-Z0-9]{4}", message="The length should be at least 4")
    private String pwd;

    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
}
```

For the controller class, write the following code.

```
import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class StudentController {
    @RequestMapping("/hello")
    public String show(Model md)
    {
        md.addAttribute("stu", new Student());
        return "vp";
    }
    @RequestMapping("/helloagain")
    public String subfrm(@Valid @ModelAttribute("stu") Student s BindingResult br)
    {
        if(br.hasErrors())
        {
            return "vp";
        }
        else
        {
            return "last";
        }
    }
}
```

In the web.xml file, write the following code.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>

```

Now, for the xml file, specify the bean like this in the spring-servlet file.

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context.xsd
           http://www.springframework.org/schema/mvc
           http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <!-- Add functionality to scan component-->
    <context:component-scan base-package="com.ith" />
    <!-- Add functionality for validation, formatting, and conversion -->
    <mvc:annotation-driven/>
    <!--Specify the view resolver in Spring MVC -->
    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
</beans>

```

Now, for the initial request, create an index.jsp page and add the following code.

```

index.jsp
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<body>
<a href="hello">Hit This Link To Go Forward!</a>
</body>
</html>

```

To show the view components, create a vp.jsp page and add the following.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<style>
.error{color:blue}
</style>
</head>
<body>
<form:form action="hiuser" modelAttribute="stu">
Student Name: <form:input path="name"/> <br><br>
Password(*) : <form:password path="pass"/>
<form:errors path="pass" cssClass="error"/><br><br>
<input type="submit" value="submit">
</form:form>
</body>
</html>
```

Finally, create one more, last.jsp and add the following.

```
<html>
<body>
Username: ${stu.sname} <br><br>
Password: ${stu.pwd}
</body>
</html>
```

When the application is run, a click leads the user to a new screen which takes the username credential. If the password is shorter than 4 characters, then there is a warning text for it. On typing the right password, the application saves it and proceeds.

20.17 | Validation with Numbers

In Spring MVC, we can use validation of user input with a set of numbers. In order to utilize the validation for regular expression, you have to use @Min annotation and @Max annotation. The input has to be more than its value for the former, whereas the input has to lesser than its value for the latter.

Add dependencies in the pom.xml file.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-webmvc</artifactId>
<version>5.1.1.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper -->
<dependency>
<groupId>org.apache.tomcat</groupId>
<artifactId>tomcat-jasper</artifactId>
<version>9.0.12</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
<groupId>javax.servlet</groupId>
```

```

<artifactId>servlet-api</artifactId>
<version>3.0-alpha-1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator -->
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>

```

Now make the Student.java class which is our bean class.

```

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Size;

public class Student {
    private String name;
    @Size(min=1,message="required")
    private String pwd;

    @Min(value=50, message="A student must score 50 or more to qualify.")
    @Max(value=100, message="A student cannot score equal or less than 100.")
    private int marks;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
    public int getMarks() {
        return marks;
    }
    public void setMarks(int marks) {
        this.marks = marks;
    }
}

```

For your controller class, type the following code:

```
import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class StudentController {
    @RequestMapping("/hiuser")
    public String show(Model md)
    {
        md.addAttribute("stu", new Student());
        return "vp";
    }
    @RequestMapping("/hiagain")
    public String subfrm( @Valid @ModelAttribute("stu") Student s, BindingResult br)
    {
        if(br.hasErrors())
        {
            return "vp";
        }
        else
        {
            return "last";
        }
    }
}
```

In the web.xml file, add the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>SpringMVC</display-name>
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

Specify a bean in the XML file. Then for the request page, add the following in the index.jsp.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<body>
<a href="hiuser">Hit this link!</a>
</body>
</html>
```

Now, to generate the view components, write the following:

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<style>
.error{color:blue}
</style>
</head>
<body>
<form:form action="hiagain" modelAttribute="student">
Name: <form:input path="name"/> <br><br>
Password: <form:password path="pass"/>
<form:errors path="pass" cssClass="error"/><br><br>
Marks: <form:input path="marks"/>
<form:errors path="marks" cssClass="error"/><br><br>
<input type="submit" value="submit">
</form:form>
</body>
</html>
```

In the last.jsp, type the following code:

```
<html>
<body>
Username: ${param.name} <br>
Password: ${param.pwd} <br>
Age: ${param.marks } <br>
</body>
</html>
```

This code would ensure that only students with marks in the range of 50–100 can submit their details.

Summary

Spring MVC remains one of the leading Java frameworks in the industry today. If you plan to learn web development for educational purposes or to begin a career, then it is an excellent choice. From the government sector to the banking sector, Java Spring MVC is used in a wide range of industries, especially in the enterprises.

In this chapter, we have learned the following concepts:

1. Spring Framework and Spring MVC.
2. Dependency Injection, Inversion of Control, and aspect-oriented programming.
3. Spring architecture including core module, web module, Bean module, and context module.
4. Bean scope such as singleton, prototype, and session.
5. Spring MVC concepts.
6. DispatcherServlet, Context Hierarchy, Special Bean Types, etc.
7. Spring Bean Processing, Interception, Chain of Resolver.
8. View Resolution, Multiple Controller, etc.
9. Various examples to understand concepts such as Model Interface and RequestParam.

In Chapter 21, we will learn about a popular ORM tool known as Hibernate. We will learn how to use ORM tools such as annotations, inheritance mapping, and class to table mapping.

Multiple-Choice Questions

1. Which one of the following components intercepts all requests in a Spring MVC application?
 - (a) DispatcherServlet
 - (b) ControllerServlet
 - (c) FilterDispatcher
 - (d) None of the above
2. Which one of the following components is utilized to map a request to a method of a controller?
 - (a) URL Mapper
 - (b) RequestResolver
 - (c) RequestMapper
 - (d) RequestMapping
3. If we want to apply custom validation, which one of the following interfaces is required to be implemented?
 - (a) Validatable
4. Is it possible to generate multiple controllers in Spring MVC at the same time?
 - (a) Yes
 - (b) No
5. _____ uses the DispatcherServlet for invoking handlers which are mapped with a request.
 - (a) HandlerMapping
 - (b) HandlerAdapter
 - (c) LocalResolver
 - (d) FlashMapManager

Review Questions

1. How do you define Spring Framework?
2. What are the core modules of Spring Framework?
3. What is Dependency Injection?
4. What are the benefits of using Dependency Injection?
5. What is Inversion of Control?
6. How does Spring use Inversion of Control? What are the benefits?
7. What is aspect-oriented programming? How do we use it?
8. What are Bean Scopes?
9. What is the default Bean Scope?
10. When do we use Prototype Scope?
11. What is Spring MVC?
12. What is Dispatcher Servlet?
13. How does Autowiring work?
14. How does constructor-based Dependency Injection work?
15. How does setter-based Dependency Injection work?
16. How does field-based Dependency Injection work?
17. Which type of Dependency Injection is better to use? Why?
18. What types of advice are there? Explain in detail.
19. What is Pointcut?
20. What is Join Point?
21. What is the difference between Pointcut and Join Point?
22. What is Target object?
23. What is AOP Proxy?
24. What modules are there in the Data Access/Integration layer?
25. What is the use of Messaging Module?
26. What is context hierarchy?
27. What is Chain of Resolvers? How does it work?
28. What is view resolution?
29. What is the difference between @RequestParam and @PathVariable?

Exercises

1. Setup an environment either using Eclipse or Spring Tool Suite and create a Spring MVC project. Create three views, first to capture student data for registration using FormTag Library, second to display data and third home page to show a school page.
2. Extend this first exercise to add validations on the registration form.

Project Idea

Create a Hotel Booking application using Spring MVC. This app should have frontend view to search for hotels based on users' criteria such as star ratings, reviews, and price. And a feature to sort results based on price or ratings. It should also

have front end view for admin to add hotel information. Use HTML, CSS, JQuery, and Bootstrap for the front end and use Spring MVC on the back end side.

Recommended Readings

1. Iuliana Cosmina, Rob Harrop, Chris Schaefer, and Clarence Ho. 2017. *Pro Spring 5: An In-Depth Guide to the Spring Framework and its Tools*. Apress: New York
2. Ranga Karanam. 2017. *Mastering Spring 5.0*. Packt: Birmingham
3. Tejaswini Mandar Jog. 2017. *Learning Spring 5.0: Build enterprise grade applications using Spring MVC, ORM Hibernate and RESTful APIs*. Packt: Birmingham

Introduction to Hibernate

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- ORM tools.
- Hibernate and how to use Hibernate.
- How to configure database and connect to it.
- Various useful annotations.
- Inheritance mapping.
- How to map Class and Table.
- How to map with set.
- Transactions and caching.
- Hibernate Query Language.
- All about named query.
- Spring Integration.

21.1 | Introduction

Hibernate is one of the most popular and common technologies in Java related projects. It is a framework which acts as an intermediary between a Java application and a database. All the low-level complex work associated with SQL is addressed by Hibernate. Through the addition of Hibernate, the number of lines of code for JDBC can be greatly decreased. The key offering of Hibernate is object-relational mapping (ORM).

What is ORM? As a developer, you have to specify in Hibernate about how an object in your application is mapped to the stored data in the DB. This means that it would map your Java class to any table in the database.

For example, if there is a Java class Employee with four fields:

1. id(int)
2. name(String)
3. designation(String)
4. department(String)

And you have a database table known as Employee.

1. id (INT)
2. name(VARCHAR(45))
3. designation(VARCHAR(45))
4. department(VARCHAR(45))

Then what Hibernate does is that it ensures that this class and table are mapped appropriately. For instance, it can use the object to change the designation of an employee in the table. To generate a Java object in Hibernate, you can write the following code.

```
Employee emp = new Employee ("Albert", "developer", "IT");
```

In order to save it in the DB, you can write the following.

```
int infoId = (Integer) session.save(emp);
```

By doing this, Hibernate is simply running the SQL insert query on the back and saving the data in the DB table. To get or retrieve an object through Hibernate, you can write the following.

```
Employee emp = session.get(Employee.class, infoId);
```

In this code the `session.get` method searches for the class of the object and also takes the primary id of the object (i.e., "infoId"). On the back, Hibernate processes this information by going to the DB and looking for a column with this primary ID.

Now, what to do if you require all of the "Employee" objects? This can be done through Hibernate's support for query. To do this, you can write the following.

```
Query q = session.createQuery("from Employee");
List<Employee> employees = query.list();
```

What the above code does is that it goes in the database table with the name "Employee" and then list all its entries. This is Hibernate Query Language (HQL) at work which would use explain later in our tutorial.

QUICK CHALLENGE

Create a chart to show the differences between JDBC approach and Hibernate approach. Also list the advantages and disadvantages of using both.

21.2 | Architecture



Hibernate uses a layered architecture as shown in Figure 21.1. It allows a user to work without explicitly knowing the underlying APIs. Hibernate takes advantage of the configuration and database data which is used to offer persistence services to the application. Hibernate uses JNDI, JDBC, JTA, Java Naming, and other existing Java APIs. It is used to take advantage of a basic abstraction level for relational databases. This means that it is supported by most of the databases.

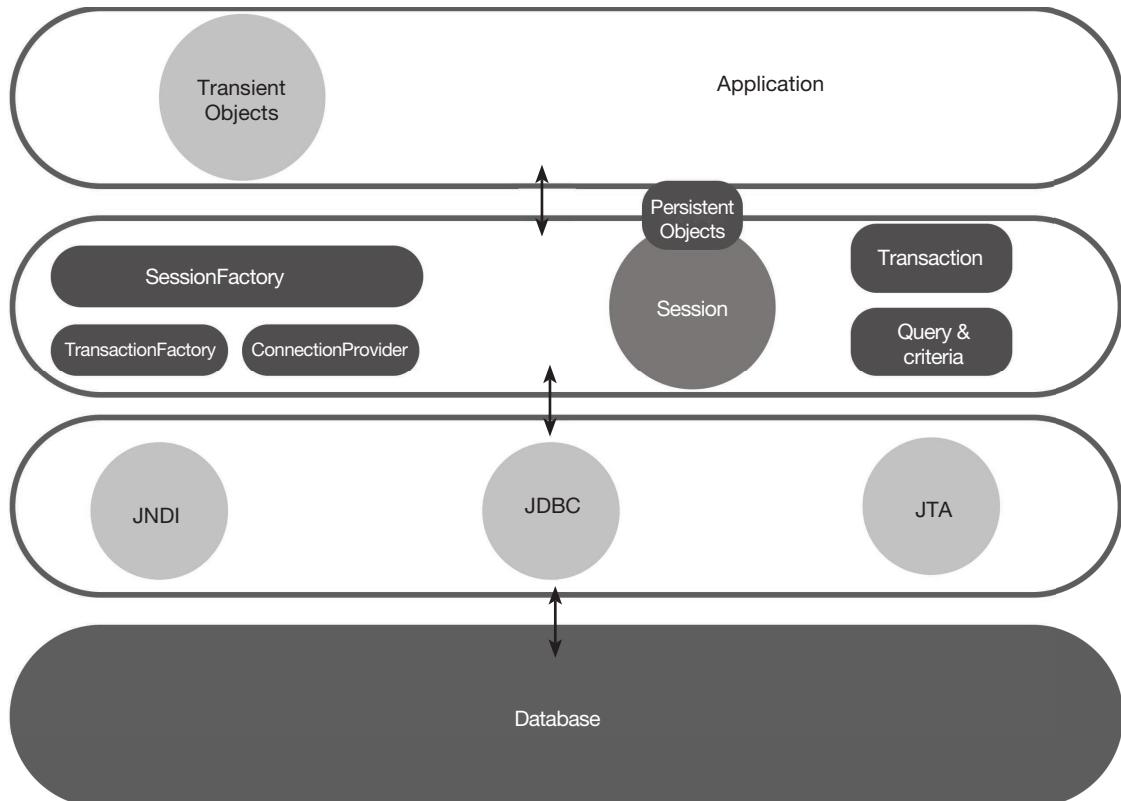


Figure 21.1 Hibernate architecture.

21.2.1 Configuration Object

The first Hibernate object in a Hibernate application is called *configuration object*. It is generated during application initialization. It has details of properties and configuration file. This object has two main components:

1. **Database connection:** This is managed by multiple configuration files. These files are hibernate.cfg.xml and hibernate.properties.
2. **Class mapping setup:** This generates the connection between the DB tables and the Java classes.



How can you connect multiple ORM tools to Spring MVC?

21.2.2 Session

To establish a physical connection with a database, you need a session. The object of session is lightweight. It is used every time a DB interaction is required. Session objects are used to retrieve persistent objects. The session objects are not allowed to remain open for long periods of time as they are not deemed thread safe. Hence, they are created and destroyed according to the application requirements. The main objective of the session object is to perform read, create, and remove operations on related mapped entity classes instances.

21.2.3 Methods

1. **Transaction beginTransaction():** It starts a unit of work and forwards the relevant Transaction object.
2. **Void cancelQuery():** It terminates the current query execution.
3. **void clear():** It fully clears the session.
4. **connection close():** It ends the session either by cleaning or releasing the connections related to JDBC.
5. **criteria createCriteria(Class persistentClass):** It generates a Criteria instance for a superclass of a specified entity class.
6. **CriteriaCreateCriteria (String entityName):** It generates a fresh Criteria instance to help with the entity name.
7. **Serializable getIdentifier (Object object) :** It is used for a given entity's identifier value.



What is the use of Transaction Management in Hibernate?

21.2.4 Hibernate Criteria

HQL is mostly used to query the database and receive the results. Usually, it is not preferred for deleting or updating values as then it means that the developer has to manage the table associations.

Hibernate Criteria API offers an object-oriented approach, which can be used to query the DB and get the output. It is important to note that the Hibernate Criteria query cannot be used to delete or update Data Definition Language (DDL) statements or other queries. Instead, it is only used for getting DB results through the use of object-oriented model. This API allows the following:

1. Hibernate Criteria API offers Projection, which can be used for min(), max(), sum(), and other aggregate functions.
2. It uses “ProjectList” for getting the specified columns.
3. Hibernate Criteria is used with join queries where it joins multiple tables.
4. Results with conditions can also be fetched by the Hibernate Criteria API.
5. It also offers the addOrder() for ordering the output.



List the differences between HQL and SQL languages.

21.2.5 Cache

Hibernate Cache is valuable in providing quick performance for applications. It helps to decrease DB queries and thus decreases the application throughput. There are three types of Cache.

1. **First level cache:** This cache is linked with the session object. By default, it is enabled and cannot be turned off. Objects which are cache through such a session are invisible for other sessions. After a session is terminate, all the cache objects are removed.
2. **Second level cache:** By default, it is disabled. To enable it, you have to modify the configuration. Infinispan and EHCache offer the Hibernate second level cache.
3. **Query cache:** Hibernate is used also for caching a query's result set. It does not cache entities' state cache. It is used for caching only the identifier values and the value type results. Hence, it is used with the second-level cache.



How will application behave if no Cache is used?

21.3 | Installation and Configuration



In order to use hibernate, we need to first install database, configure and setup database. In the following sections, we will be installing database, testing database connection using JDBC, and configuring hibernate.

21.3.1 Database

In this chapter, we are using MySQL database for the Hibernate example. Install the latest MySQL Community Server. When you are done installing it, go to Start and search for "Workbench". Open it and connect it to the server instance.

In MySQL, we have created a user "hbstudent", which contains the following columns.

1. Id
2. first_name
3. last_name
4. email

QUICK CHALLENGE

List all the databases that can work with Hibernate.

21.3.2 Eclipse

In Eclipse, create a new "Java Project", we have named ours as "hib". When the project is generated, right click on it and then go to New → Folder. Name the folder as "lib". This folder would be used to store JDBC and Hibernate files. Now, go to Hibernate's official website. Check the latest Hibernate ORM release and download it. After extracting the folder, go in the "required" folder and copy all the files. Come back to Eclipse and paste it in the "lib" folder.

Afterward, open this website and download the latest Connector. After extracting it, copy the JAR file (named like mysql-connector-java) and then paste it in the "lib" folder.

Finally, right click on the project and go to → Properties → Java Build Path → Libraries. Choose "Add Jars" and go to your project and highlight all recently added JAR files. Apply the changes and close the Properties.

21.3.3 JDBC

Before moving forward, it is important to test Java Database Connectivity (JDBC). Create a sample package in the project (we have named it "com.hib.jdbc") and then add a class in it with the following code.

```

import java.sql.Connection;
import java.sql.DriverManager;
public class exampleJDBC {
    public static void main(String args[]) {
        String jdbcUrl = "jdbc:mysql://localhost:3306/hb_student_tracker?useSSL=false";
        String user = "hbstudent";
        String pass = "hbstudent";
        try {
            System.out.println("Attempting to connect to DB: " + jdbcUrl);
            Connection myConn = DriverManager.getConnection(jdbcUrl, user, pass);
            System.out.println("Congratulations! The Connection Is Successful!");
        }
        catch (Exception exc) {
            exc.printStackTrace();
        }
    }
}

```

If everything is done correctly, then we get the following output.

```

Attempting to connect to DB:
jdbc:mysql://localhost:3306/hb_student_tracker?useSSL=false
Congratulations! The Connection Is Successful!

```

Till now, we have set up and configured everything required to begin with Hibernate. Now to configure Hibernate, we must first have to create a configuration file. This file instructs Hibernate about the information required to establish a connection with the database. It would contain the JDBC URL, id, password, and other pieces of information. Right click on the “src” of the project and then create a new file with the name “hibernate.cfg.xml”. Copy the following code in the file.

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- JDBC Database connection settings -->
        <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/hb_student_tracker?
useSSL=false&serverTimezone=UTC</property>
        <property name="connection.username">hbstudent</property>
        <property name="connection.password">hbstudent</property>
        <!-- JDBC connection pool settings ... using built-in test pool -->
        <property name="connection.pool_size">1</property>
        <!-- Select our SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <!-- Echo the SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Set the current session context -->
        <property name="current_session_context_class">thread</property>
    </session-factory>
</hibernate-configuration>

```

21.3.4 Annotations

In Hibernate, there is a term known as “Entity Class”. It is just a plain old Java object (POJO) which is mapped with a table in the database. This mapping can be done through two options. You can either use the older approach to generate XML configuration files or you can adopt the modern approach to use Annotations.

In Annotations, you have to first map a Java class to a table in the DB and then map its fields to the columns of that DB. To work with Annotations, create a new package in your project and a “Student” class in it. Add the following code in the class.

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="student")
public class Student {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="first_name")
    private String firstName;

    @Column(name="last_name")
    private String lastName;

    @Column(name="email")
    private String email;

    public Student() { }

    public Student(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastname() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    @Override
    public String toString() {
        return "Student [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName
+ ", email=" + email + "]";
    }
}
```

**QUICK
CHALLENGE**

Create a list of all the important annotations.

21.4 | Java Objects in Hibernate

In Hibernate, we are going to use a class called SessionFactory. This is the class which goes through the configuration file. It is also responsible for the generation of the session objects. Session factory is generated once and is then used several times in the application. When this class is generated, it then manufactures “sessions”.

The “session” class is the wrapper class for JDBC. It is this object which is required for the storage and retrieval of objects. Bear in mind that it has a short lifespan.

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import com.hib.hibernate.demo.entity.Student;
public class CreateStudentDemo {
    public static void main(String[] args) {
        SessionFactory factory = new Configuration().configure("hibernate.cfg.xml") .
addAnnotatedClass(Student.class).buildSessionFactory();
        Session session = factory.getCurrentSession();
        try {
            System.out.println("Generate an object for a student");
            Student tempStudent = new Student("Robert", "William", "robw123@abc.com");
            session.beginTransaction();
            System.out.println("the student object is being saved");
            session.save(tempStudent);
            session.getTransaction().commit();
            System.out.println("Completed!");
        }
        finally {
            factory.close();
        }
    }
}
```

Now, open your MySQL table and you will see a new row in your table. Change the values in the above code and run it again. Now you will see a new record in the table.



How can Hibernate manage database security?

21.4.1 Primary Key

Before going further, you must have a basic understanding of primary key (PK). PK is a DB component which is applicable on a column in a table. The main purpose of putting PK on a column is to establish a relationship in relational database. Bear in mind that PK is always unique; this means that you can only assign those columns as PK which cannot have repeated values. For instance, in an employee table, the employee numbers which are unique can be used as PK. It is also important to note that it cannot store NULL values. For instance, if there is an employee “Josh” then to update its data, its employee number “04” can be used in queries where no other employee carries the same employee number. To assign a column as PK, simply write the following line in the SQL query.

Primary Key ("emp_no")

In Java, you can use GenerationType.AUTO to choose any suitable technique for a specific DB to generate an ID. By using the GenerationType.Identity, primary keys can be assigned with respect to the identity column. The GenerationType.SEQUENCE class processes primary keys according to a DB sequence. Finally, the GenerationType.TABLE processes primary keys with respect to an underlying table in the DB for maintaining uniqueness.

Go to your MySQL Workbench and right click on the table → Alter Table. You can see your columns and the primary key. In our example, we have turned on auto-increment so your AI column will also be ticked.

21.5 | Inheritance Mapping



There are three types of inheritance mapping techniques in Hibernate.

21.5.1 Table Per Hierarchy

In this mapping, you require `@Inheritance(strategy=InheritanceType.Single_TABLE)`, `@DiscriminatorColumn`, and `@DiscriminatorValue` annotations. This mapping requires only a single table. One more column is generated in the DB's table for calling the class. This table is referred to as the discriminator column. For example, first generate a persistent class which embodies inheritance. To do this, we have produced three classes. First, we have a `Student.java` class.

```
package com.hib.example;
import javax.persistence.*;
@Entity
@Table(name = "Jones")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="student")
public class Student {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "id")
    private int id;
    @Column(name = "name")
    private String name;
    //setters and getters
}
Our second class is the "fulltime_student.java" class.
package com.hib.example;
import javax.persistence.*;

@Entity
@DiscriminatorValue("fulltimestudent")
public class FulltimeStudent extends Student{

    @Column(name="marks")
    private float marks;

    @Column(name="coursenumber")
    private int coursenumber;

    //setters and getters
}
```

Our final class is the `parttime_student.java` class.

Now, in the project go in the source of the `pom.xml` file and add the following dependencies. Make sure that they are placed within the `<dependencies>` tag.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>
```

Now, you have to open your hibernate.cfg.xml file and add the list of entity classes.

```
?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-5.3.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.ith.mypackage.Student"/>
        <mapping class="com.ith.mypackage.Fulltime_Student"/>
        <mapping class="com.ith.mypackage.Parttime_Student"/>
    </session-factory>
</hibernate-configuration>
```

However, in order to store persistent object, you will require a class. To do this, create a “StoreTest” Java class.

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreTest {
    public static void main(String args[])
    {
        StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure
        ("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
        SessionFactory factory=meta.getSessionFactoryBuilder().build();
        Session session=factory.openSession();
        Transaction t=session.beginTransaction();

        Employee e1=new Employee();
        e1.setName("Gaurav Chawla");

        Regular_Employee e2=new Regular_Employee();
        e2.setName("Vivek Kumar");
        e2.setSalary(50000);
        e2.setBonus(5);

        Contract_Employee e3=new Contract_Employee();
        e3.setName("Arjun Kumar");
        e3.setPay_per_hour(1000);
        e3.setContract_duration("15 hours");

        session.persist(e1);
        session.persist(e2);
        session.persist(e3);

        t.commit();
        session.close();
        System.out.println("success");
    }
}
```

Run this class and you will see the list of students with their names and other details.

21.5.2 Table Per Concrete Class

In Table per Concrete class, a table is generated for each class. This means that in the DB, there is no table which has nullable values. On the other hand, this gives rise to duplication of columns.

In such a mapping, the `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)` annotation is used in the parent class, while the `@AttributeOverrides` annotation is used in any of the subclasses. The former annotation defines the implementation of Table per Concrete class technique while the latter annotation specifies that the attributes of the parent class would be overridden.

First, generate persistent classes for inheritance.

```
import javax.persistence.*;

@Entity
@Table(name = "employee")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Employee {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)

    @Column(name = "eid")
    private int eid;

    @Column(name = "e_name")
    private String ename;

    //setters and getters
}
```

Now, generate a class for permanent employees.

```
import javax.persistence.*;

@Entity
@Table(name="permanentemployees102")
@AttributeOverrides({
    @AttributeOverride(name="eid", column=@Column(name="eid")),
    @AttributeOverride(name="ename", column=@Column(name="ename"))
})
public class PermanentEmployee extends Employee{

    @Column(name="pay")
    private float pay;

    @Column(name="leaves")
    private int leaves;

    //setters and getters
}
```

For temporary employees, generate the following class.

```

import javax.persistence.*;
@Entity
@Table(name="temporaryemployees")
@AttributeOverrides({
    @AttributeOverride(name="eid", column=@Column(name="eid")),
    @AttributeOverride(name="ename", column=@Column(name="ename"))
})
public class temporary_Employee extends Employee{
    @Column(name="hourly_wage")
    private float hourly_wage;

    @Column(name="services_type ")
    private String services_type;

    public float gethourly_wage() {
        return hourly_wage;
    }
    public void sethourly_wage(float hourlywage) {
        hourly_wage = hourlywage;
    }
    public String getservices_type() {
        return services_type;
    }
    public void setservices_type(String servicestype) {
        services_type = servicestype;
    }
}

```

Now, in the project go in the source of the pom.xml file and add the following dependencies. Make sure that they are placed within the <dependencies> tag.

```

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>

In the hibernate.cfg.xml file, apply the following mapping.
?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<!-- Generated by MyEclipse Hibernate Tools. -->
<hibernate-configuration>
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">abc</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.ith.examples.Employee"/>
        <mapping class="com.ith.examples.PermanentEmployee"/>
        <mapping class="com.ith.examples.Temporary_Employee"/>
    </session-factory>
</hibernate-configuration>

```

Now generate a class `StoreData` which can save all these objects related to employee tables.

```

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {
        StandardServiceRegistry reg=new StandardServiceRegistryBuilder().configure
        ("hibernate.cfg.xml").build();
        Metadata meta=new MetadataSources(reg).getMetadataBuilder().build();
        SessionFactory factory=meta.getSessionFactoryBuilder().build();
        Session session=factory.openSession();
        Transaction trans=session.beginTransaction();

        Employee first=new Employee();
        first.setename("Jack");

        PermanentEmployee second=new PermanentEmployee();
        second.setename("Jones");
        second.setpay(4000);
        second.setleaves(8);

        Temporary_Employee third=new Temporary_Employee();
        third.setename("Bill");
        third.sethourly_wage(2000);
        third.setservices_type("IT");

        session.persist(first);
        session.persist(second);
        session.persist(third);

        trans.commit();
        session.close();
        System.out.println("We have completed the operation successfully");
    }
}

```

21.5.3 Table Per Subclass

In table per subclass, tables are generated as persistent classes though they act as primary and foreign keys. This means that it is not possible to have duplicate columns in the table.

In this technique, you require the parent class to have “`@Inheritance(strategy=InheritanceType.JOINED)`” and the subclasses to have `@PrimaryKeyJoinColumn` annotations.

First, we have to generate the persistent classes for inheritance. The first “Employee” class contains the following code.

```

import javax.persistence.*;
@Entity
@Table(name = "employee105")
@Inheritance(strategy=InheritanceType.JOINED)
public class Employee {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "eid")
    private int eid;

    @Column(name = "ename")
    private String ename;

    //setters and getters
}

```

In the second class, “PermanentEmployee” we have the following code.

```
import javax.persistence.*;
@Entity
@Table(name="permanentemployee105")
@PrimaryKeyJoinColumn(name="eid")
public class PermanentEmployee extends Employee{

    @Column(name="pay")
    private float pay;
    @Column(name="leaves")
    private int leaves;

    //setters and getters
}
```

In the third class, we have the following code.

```
import javax.persistence.*;
@Entity
@Table(name="temporaryemployee105")
@PrimaryKeyJoinColumn(name="eid")
public class Temporary_Employee extends Employee{

    @Column(name="hourly_wage")
    private float hourly_wage;

    @Column(name="services_type")
    private String services_type;

    //setters and getters
}
```

Now, in the project go in the source of the pom.xml file and add the following dependencies. Make sure that they are placed within the <dependencies> tag.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.3.1.Final</version>
</dependency>

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>
```

For the configuration file, do the following.

```

?xml version='1.0' encoding='UTF-8'?
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<!-- Generated by MyEclipse Hibernate Tools. -->
<hibernate-configuration>
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">abc</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

        <mapping class="com.ith.examples.Employee"/>
        <mapping class="com.ith.examples.PermanentEmployee"/>
        <mapping class="com.ith.examples.Temporary_Employee"/>
    </session-factory>
</hibernate-configuration>

```

Now in the final StoreData class, write the following code.

```

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {
        StandardServiceRegistry reg=new StandardServiceRegistryBuilder()
configure("hibernate.cfg.xml").build();
        Metadata meta=new MetadataSources(reg).getMetadataBuilder().build();
        SessionFactory factory=meta.getSessionFactoryBuilder().build();
        Session session=factory.openSession();
        Transaction trans=session.beginTransaction();
        Employee first=new Employee();
        first.setename("Ryan");

        PermanentEmployee second=new PermanentEmployee();
        second.setename("James");
        second.setpay(7000);
        second.setleaves(4);

        Temporary_Employee third=new Temporary_Employee();
        third.setename("Luke");
        third.sethourly_wage(5000);
        third.setservices_type("accounting");

        session.persist(first);
        session.persist(second);
        session.persist(third);

        trans.commit();
        session.close();
        System.out.println("We have completed the operation successfully");
    }
}

```



21.6 | Collection Mapping

Hibernate supports mapping of collection elements in the persistent class. Their types include list, set, map, and collection. The persistent class has to be specified like the following:

```
import java.util.List;

public class problem {
    private int id;
    private String pname;
    private List<String> solutions;

    //getters and setters
}
```

There are numerous sub-elements in the <class> elements for mapping collections.

21.6.1 Mapping List

When the persistent class contains objects in the “list” format, they can be through either annotation or file. For the above class, we can write.

```
<class name="com.ihrt.problem" table="p100">
    <id name="id">
        <generator class="increment"></generator>
    </id>
    <property name="pname"></property>

    <list name="solutions" table="sol100">
        <key column="pid"></key>
        <index column="type"></index>
        <element column="solution" type="string"></element>
    </list>
</class>
```

Here, the <key> element is required to specify the table’s foreign key with respect to the problem class identifier. To specify the type, the <index> element is required. To specify the collection’s element, <element> is used.

In case collection saves objects of another class, it is necessary to specify the types of relationship one-to-many or many-to-many. In such instances, the persistent class resembles the following.

```
import java.util.List;

public class problem {
    private int id;
    private String pname;
    private List<Solution> solutions;

    //getters and setters
}
import java.util.List;
public class Solution {
    private int id;
    private String solution;
    private String posterName;
    //getters and setters
}
```

The mapping file would be similar to the following.

```
<class name="com.ith.problem" table="p100">
    <id name="id">
        <generator class="increment"></generator>
    </id>
    <property name="pname"></property>
    <list name="solutions" >
        <key column="pid"></key>
        <index column="type"></index>
        <one-to-many class="com.ith.problem"/>
    </list>
</class>
```

The one-to-many relationship here means that a single problem can have multiple solutions.

21.6.2 Key Element

As already explained, the key element is required to place foreign keys in tables. These tables are joined. By default, FK element is nullable. For non-nullable FK, it is important to define attributes which are not null like.

```
<key column="pid" not-null="true"></key>
```

21.6.3 Collection Mapping with Lists

Generate a persistent class.

```
import java.util.List;
public class Problem {
    private int id;
    private String pname;
    private List<String> solutions;
    //getters and setters
}
```

Now create a configuration file named as problem.hbm.xml and apply the following.

```
?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.ith.problem" table="p100">
        <id name="id">
            <generator class="increment"></generator>
        </id>
        <property name="pname"></property>
        <list name="solutions" table="sol100">
            <key column="pid"></key>
            <index column="type"></index>
            <element column="solution" type="string"></element>
        </list>
    </class>
</hibernate-mapping>
```

In the configuration file, apply the following.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">user1</property>
        <property name="connection.password">abc</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="problem.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

To save the data of the problem class, use the StoreData class.

```
import java.util.ArrayList;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {
        StandardServiceRegistry reg=new StandardServiceRegistryBuilder()
            .configure("hibernate.cfg.xml").build();
        Metadata meta=new MetadataSources(reg).getMetadataBuilder().build();

        SessionFactory fac=meta.getSessionFactoryBuilder().build();
        Session session=fac.openSession();

        Transaction ts=session.beginTransaction();

        ArrayList<String> l1=new ArrayList<String>();
        l1.add("Spring is an ecosystem");
        l1.add("Spring is used to create enterprise web applications");

        ArrayList<String> l2=new ArrayList<String>();
        l2.add("Hibernate is an intermediary between database and applications");
        l2.add("Hibernate is used for managing the dependencies of the application");

        Problem p1=new Problem();
        p1.setPname("What is Spring Framework and why is it used?");
        p1.setSolutions(l1);

        Problem p2=new Problem();
        p2.setPname("What is Hibernate and why is it used?");
        p2.setSolutions(l2);

        session.persist(p1);
        session.persist(p2);

        ts.commit();
        session.close();
        System.out.println("The operation was successful");
    }
}
```

21.6.4 Mapping with Set

When there is a set object in the persistent class, then mapping can be done through the mapping file's set element. This element does not need an index element. Set differs from the list as it only contains unique values. Generate a persistent class named "Problem" which has the properties such as pname and solutions.

```
import java.util.Set;
public class Problem {
    private int id;
    private String pname;
    private Set<String> solutions;
    //getters and setters
}
```

Now create a configuration file named problem.hbm.xml and apply the following.

```
?xml version='1.0' encoding='UTF-8'?
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.ith.problem" table="p1003">
        <id name="id">
            <generator class="increment"></generator>
        </id>
        <property name="pname"></property>
        <set name="solutions" table="sol1003">
            <key column="pid"></key>
            <index column="type"></index>
            <element column="solution" type="string"></element>
        </set>
    </class>
</hibernate-mapping>
```

In the configuration file, apply the following.

```
<?xml version='1.0' encoding='UTF-8'?
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">user1</property>
        <property name="connection.password">abc</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping resource="problem.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

To save the data of the problem class, use the StoreData class.

```

import java.util.HashSet;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
    public static void main(String[] args) {
        StandardServiceRegistry reg=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta=new MetadataSources(reg).getMetadataBuilder().build();
        SessionFactory fac=meta.getSessionFactoryBuilder().build();
        Session session=fac.openSession();
        Transaction ts=session.beginTransaction();

        HashSet<String> s1=new HashSet<String>();
        s1.add("Linked list is a data structure");
        s1.add("Linked list is good for memory purposes");

        HashSet<String> s2=new HashSet<String>();
        s2.add("In stack data structure, the last data to be inserted is processed first");
        s2.add("Stack data structure is based upon the real world concept of stacks like book stacks");

        Problem p1=new Problem();
        p1.setPname("What is a linked list and why is it used?");
        p1.setSolutions(s1);

        Problem p2=new Problem();
        p2.setPname("What is Stack data structure and where did the idea come from?");
        p2.setSolutions(s2);

        session.persist(p1);
        session.persist(p2);

        ts.commit();
        session.close();
        System.out.println("The operation was successful");
    }
}

```

21.7 | Mapping with Map

With Hibernate, it is possible to “map” elements of Map along with relational database management systems. The map is a collection which uses indexes on its back. For the problem class, write the following code.

```

import java.util.Map;

public class Problem {
    private int id;
    private String pname,username;
    private Map<String, String> solutions;

    public Problem() {}
    public Problem(String pname, String username, Map<String, String> solutions) {
        super();
        this.pname = pname;
        this.username = username;
        this.solutions = solutions;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getPname() {
        return pname;
    }
    public void setPname(String pname) {
        this.pname = pname;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public Map<String, String> getSolutions() {
        return solutions;
    }
    public void setSolutions(Map<String, String> solutions) {
        this.solutions = solutions;
    }
}

```

For the problem.hbm.xml file, apply the following.

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">
<hibernate-mapping>
    <class name="com.ith.problem" table="problem800">
        <id name="id">
            <generator class="native"></generator>
        </id>
        <property name="name"></property>
        <property name="username"></property>

        <map name="solutions" table="solution800" cascade="all">
            <key column="problemid"></key>
            <index column="solution" type="string"></index>
            <element column="username" type="string"></element>
        </map>
    </class>
</hibernate-mapping>

```

For the hibernate.cfg.xml file, apply the following.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">user1</property>
    <property name="connection.password">abc</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
  </property>
    <mapping resource="problem.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

To save the data and run the application, generate the “StoreData” class and add the following.

```
Import java.util.HashMap;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class StoreData {
  public static void main(String[] args) {
    StandardServiceRegistry reg=new StandardServiceRegistryBuilder().configure
("hibernate.cfg.xml").build();
    Metadata meta=new MetadataSources(reg).getMetadataBuilder().build();

    SessionFactory fac=meta.getSessionFactoryBuilder().build();
    Session session=fac.openSession();

    Transaction ts=session.beginTransaction();
    HashMap<String, String> m1=new HashMap<String, String>();
    m1.put("Linked list is a data structure", "Adam");
    m1.put("Linked list is good for memory purposes", "Keith");

    HashMap<String, String> m2=new HashMap<String, String>();
    m2.put("In stack data structure, the last data to be inserted is processed first",
"Daniel");
    m2.put("Stack data structure is based upon the real world concept of stacks like
book stacks", "Daniel");
    problem p1=new problem("What is Linked List and why is it used?", "Scully", m1);
    problem p2=new problem("What is Stack?", "Simon", m2);
    session.persist(p1);
    session.persist(p2);

    ts.commit();
    session.close();
    System.out.println("The operation was successful");
  }
}
```

21.7.1 One-to-Many

When the persistent class contains an object in the form of list with the reference for entity, then mapping is performed through the one-to-many association. In these scenarios, one problem can have multiple solutions where each problem has its own details. Such a persistent class would have the following code.

```
import java.util.List;
public class Problem {
    private int id;
    private String pname; // name of the problem
    private List<Solution> solutions;
}
```

On the other hand, the Solution class can contain details like the name of the solution, poster etc.

```
public class Solution {
    private int id;
    private String solutionname;
    private String poster; // the one who posted the solution
}
```

In the Problem class, there are entity's references in the list object for which you required to apply one-to-many association.

In the mapping file, apply the following settings.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

<hibernate-mapping>
    <class name="com.ith.Problem" table="p601">
        <id name="id">
            <generator class="increment"></generator>
        </id>
        <property name="pname"></property>

        <list name="Solutions" cascade="all">
            <key column="pid"></key>
            <index column="type"></index>
            <one-to-many class="com.ith.Solution"/>
        </list>
    </class>

    <class name="com.ith.Solution" table="sol601">
        <id name="id">
            <generator class="increment"></generator>
        </id>
        <property name="solutionname"></property>
        <property name="poster"></property>
    </class>
</hibernate-mapping>
```

For the configuration file, apply the following settings.

```
?xml version='1.0' encoding='UTF-8'?
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">abc</property>
    <property name="connection.password">xyz</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="question.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

To save the data, you can generate the following class.

```
import java.util.ArrayList;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
public class StoreData {
  public static void main(String[] args) {
    StandardServiceRegistry reg= new StandardServiceRegistryBuilder().configure
("hibernate.cfg.xml").build();
    Metadata m=new MetadataSources(reg).getMetadataBuilder().build();
    SessionFactory fact=m.getSessionFactoryBuilder().build();
    Session session=fact.openSession();
    Transaction ts=session.beginTransaction();
    Solution sol1=new Solution();
    sol1.setSolutionname("HTML is a markup language");
    sol1.setPoster("Andrew");

    Solution sol2=new Solution();
    sol2.setSolutionname("HTML is a front-end technology");
    sol2.setPoster("Michael David");
    Solution sol3=new Solution();
    sol3.setSolutionname("Java is used to develop Android applications. It is also used
to create desktop and web applications.");
    sol3.setPoster("Johnny Garcia");
    Solution sol4=new Solution();
    sol4.setSolutionname("Spring Boot minimizes the amount of lines of code which is
usually required in Spring framework. Therefore, it is used for quick projects.");
    sol4.setPoster("Daniel Smith");
    ArrayList<Solution> l1=new ArrayList<Solution>();
    l1.add(sol1);
    l1.add(sol2);
    ArrayList<Solution> l2=new ArrayList<Solution>();
    l2.add(sol3);
    l2.add(sol4);
```

```

        Problem p1=new Problem();
        p1.setPname("What is HTML?");
        p1.setSolutions(l1);

        Problem p2=new Problem();
        p2.setPname("What is SpringBoot and why is this framework used?");
        p2.setSolutions(l2);

        session.persist(p1);
        session.persist(p2);

        ts.commit();
        session.close();
        System.out.println("The operation was completed successfully");
    }
}

```

21.7.2 Transaction

A transaction is used to mark a unit of work. When a single step faces failure then the complete process faces failure. Hibernate provides an interface for transaction. In Hibernate, Sessions are used to call transactions. Following are some of the methods of transactions.

1. **void begin():** Initiates a fresh transaction.
2. **void commit():** Puts a stop to the unit of work.
3. **boolean isAlive():** Scans whether the transaction is dead or alive.
4. **void setTimeout(int seconds):** It specifies a time interval when a transaction is initiated due to a call.
5. **void rollback():** Rollsbacks the transaction.

It is recommended that whenever an exception is expected in Hibernate, you should attempt at rollbacks the transaction. This strategy ensures that the resources are free. For instance, consider the following code.

```

Session ssn = null;
Transaction ts = null;
try {
    ssn = sessionFactory.openSession();
    ts = ssn.beginTransaction();
    //code for an action
    ts.commit();
} catch (Exception e) {
    e.printStackTrace();
    ts.rollback();
}
finally {
    ssn.close();
}

```

21.8 | Hibernate Query Language

Similar to the database query language SQL, Hibernate has its own query language known as Hibernate Query Language, or HQL in short. HQL is not reliant on the table name of a database. Instead, it makes use of the class name to apply its function. Therefore, it is considered to be independent of database. Hibernate is easy to learn for Java developers. Additionally, it also offers polymorphic queries.



The query interface is an OOP implementation of Hibernate's Query. Query objects can be accessed through the use of `createQuery()` method.

In order to generate data for the existing records, you can write the following code for a persistent class with the name "student". As you can observe, instead of calling a table, HQL is calling for the class through its identifier.

```
Query q = session.createQuery("from Student");//  
List l = q.list();
```

To update the details of a user, you can write the following.

```
Transaction ts = ssn.beginTransaction();  
Query q=session.createQuery("Update update User set name=:n where id=:i");  
q.setParameter("n","Buck");  
q.setParameter("i",222);  
  
int sts=q.executeUpdate();  
System.out.println(sts);  
ts.commit();
```

For deletion, consider the following example.

```
Query q=session.createQuery("delete from Student where id=305");  
q.executeUpdate();
```

You can also apply aggregate functions in HQL. For instance to check the average marks,

```
Query query =session.createQuery("select avg(marks) from Student");  
List<Integer> l=query.list();  
System.out.println(list.get(0));
```

To check the minimum marks for a student, you can write the following.

```
Query query =session.createQuery("select min(marks) from Student");
```

To check the maximum marks for a student, you can write the following.

```
Query query =session.createQuery("select max(marks) from Student");
```

To check the total number of students, you can write the following.

```
Query query =session.createQuery("select count(id) from Student");
```

To check the total marks of students, you can write the following.

```
Query query = session.createQuery("select sum(marks) from Student");
```

21.8.1 HCQL

To retrieve records according to the customized filter, Hibernate Criteria Query Language (HCQL) is used. It contains an interface for criteria which is composed of several methods to define set criteria for retrieval of information. For example, you can use HCQL to only see those students who have scored more than 80 marks. Similarly, HCQL can be used to identify only those Students who are in their final year.

To receive all the records for HCQL, you can write the following code.

```
Criteria crt=session.createCriteria(Student.class);  
List l=crt.list();  
To check only those records which exist between the 30th and 40th position, you can  
write the following.  
Criteria crt=session.createCriteria(Student.class);  
crt.setFirstResult(30);  
crt.setMaxResult(40);  
List l=crt.list();
```

To check the records on the basis of marks of students in an ascending arrangement, you can write the following.

```
Criteria crt = session.createCriteria(Student.class);  
crt.addOrder(Order.asc("marks"));  
List l = crt.list()
```

To check records of only those students who have scored more than 75, you can write the following code.

```
Criteria crt = session.createCriteria(Student.class);  
crt.add(Restrictions.gt("marks",75));  
List l = crt.list();
```

To get a specific column through projection, you can write the following piece of code.

```
Criteria crt = session.createCriteria(Student.class);  
crt.setProjection(Projections.property("department"));  
List l = crt.list();
```

21.8.2 Hibernate Named Query

The Hibernate named query is a strategy which is used to run queries through a specific name. It is similar to the DB concept of alias names. To run a single query, you can use the @NamedQuery annotation while multiple queries can be handled through @NameQueries.

For instance, consider the following code.

```

import javax.persistence.*;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@NamedQueries(
{
    @NamedQuery(
        name = "findStudentByName",
        query = "from Student s where s.name = :name"
    )
}
)

@Entity
@Table(name="st")
public class Student {
    public String toString(){return id+" "+name+" "+marks+" "+subject; }

    int id;
    String name;
    int marks;
    String subject;
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)

    //getters and setters
}

```

In the configuration file, you can save details pertaining to username, password, driver class, etc.

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">abc</property>
        <property name="connection.password">xyz</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
        <mapping class="com.ith.Student"/>
    </session-factory>
</hibernate-configuration>

```

You can use a FetchInfo class to add your named query.

```

import java.util.*;
import javax.persistence.*;
import org.hibernate.*;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchInfo {
    public static void main(String[] args) {
        StandardServiceRegistry reg=new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata m=new MetadataSources(reg).getMetadataBuilder().build();
        SessionFactory factory=m.getSessionFactoryBuilder().build();
        Session session=factory.openSession();
        TypedQuery q = session.getNamedQuery("findStudentByName");
        q.setParameter("name","Jack");
        List<Student> students=q.getResultList();
        Iterator<Student> i=students.iterator();
        while(i.hasNext()){
            Student s=i.next();
            System.out.println(s);
        }
        session.close();
    }
}

```

21.9 | Caching



To enhance an application's performance, Hibernate offers caching which uses the cache to pool an object. This is valuable when the same data requires to be processed continuously. There are two types of cache: first and second level cache.

1. The data of first level cache is stored by the Session object and is configured to be turned on by default. The application as a whole does not have access to this cache.
2. The data of second level cache is stored by the SessionFactory object. Unlike the first level cache, this cache can be accessed by the entire application.

The below example generates a persistent class for Student.

```

import javax.persistence.*;
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
@Entity
@Table(name="s101")
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
public class Student {
    @Id
    private int id;
    private String name;
    private float marks;

    public Student() {}
}

```

```

public Student(String name, float marks) {
    super();
    this.name = name;
    this.marks = marks;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public float getMarks() {
    return marks;
}
public void setMarks(float marks) {
    this.marks = marks;
}
}
}

```

In the pom.xml file, add the following dependencies.

```

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.16.Final</version>
</dependency>

<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.4.0</version>
</dependency>

<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.10.3</version>
</dependency>

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
    <version>5.2.16.Final</version>
</dependency>

```

In the configuration file, you have to specify the cache.provider_class in the property.

```
?xml version='1.0' encoding='UTF-8'?
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 5.2.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-5.2.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">abc</property>
    <property name="connection.password">xyz</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="cache.use_second_level_cache">true</property>
    <property name="cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
    <mapping class="com.ith.Student"/>
  </session-factory>
</hibernate-configuration>
```

To generate the class that can be used for the retrieval of the persistent object, consider the following code.

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class FetchInfo {
    public static void main(String[] args) {
        StandardServiceRegistry reg=new StandardServiceRegistryBuilder()
        .configure("hibernate.cfg.xml").build();
        Metadata m=new MetadataSources(reg).getMetadataBuilder().build();
        SessionFactory fact=m.getSessionFactoryBuilder().build();
        Session ssn=fact.openSession();
        Student s1=(Student)ssn.load(Student.class,202);
        System.out.println(s1.getId()+" "+s1.getName()+" "+s1.getMarks());
        ssn.close();
        Session ssn2=fact.openSession();
        Employee s2=(Employee)ssn2.load(Employee.class,202);
        System.out.println(s2.getId()+" "+s2.getName()+" "+s2.getSalary());
        ssn2.close();
    }
}
```

21.10 | Spring Integration



Spring is one of the most popular Java web frameworks in the industry. Hibernate is commonly used with Spring applications to build enterprise level web projects. Unlike previous cases where it was necessary to use the hibernate.cfg.xml file, this is not required in the case of Spring. Instead, those details would go in the file named applicationContext.xml.

A HibernateTemplate class comes up with the Spring framework, thereby eliminating the need of configuration, session, BuildSessionFactory, and transactions. For instance, without Spring, Hibernate would be required to look like the following code for a student.

```

Configuration conf=new Configuration();
conf.configure("hibernate.cfg.xml");
SessionFactory fact=conf.buildSessionFactory();
Session ssn=fact.openSession();
Transaction ts=ssn.beginTransaction();
Student s1=new Student(222,"Robert",67);
ssn.persist(s1);
ts.commit();
ssn.close();

```

On the other hand, if you use Hibernate's Template which is provided by Spring, then you can minimize the above code into only two lines.

```

Student s1=new Student(222,"Robert",67);
hibernateTemplate.save(s1);

```

For a more detailed example, use any database to generate a table.

```

CREATE TABLE "ST200"
( "ID" NUMBER(10,0) NOT NULL ENABLE,
  "NAME" VARCHAR2(255 CHAR),
  "MARKS" FLOAT(126),
  PRIMARY KEY ("ID") ENABLE
)

```

Now generate a Student class.

```

public class Student {
    private int id;
    private String name;
    private float marks;
}

```

For the mapping file,

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.ith.Student" table="ST200">
        <id name="id">
            <generator class="assigned"></generator>
        </id>
        <property name="name"></property>
        <property name="marks"></property>
    </class>
</hibernate-mapping>

```

Use a class which makes use of the `HibernateTemplate`.

```
import org.springframework.orm.hibernate3.HibernateTemplate;
import java.util.*;
public class StudentDao {
    HibernateTemplate t;
    public void setTemplate(HibernateTemplate t) {
        this.t = t;
    }
    // to store the data of student
    public void saveStudent(Student s) {
        t.save(s);
    }
    //to update the data of student
    public void updateStudent(Student s){
        t.update(s);
    }
    //to delete student
    public void deleteStudent(Student s) {
        t.delete(s);
    }
    //to get a student through id
    public Student getById(int id){
        Student s=(Student)t.get(Student.class,id);
        return s;
    }
    //to get all the students
    public List<Student> getStudents(){
        List<Student> l=new ArrayList<Student>();
        l=t.loadAll(Student.class);
        return l;
    }
}
```

In the file for `applicationContext.xml`, you have to add the details of the DB for an object named as `BasicDataSource`. The use of this object can be seen in another object, `LocalSessionFactoryBean`, which holds data related to the `HibernateProperties` and `mappingResources`. This object is required in the class of `HibernateTemplate`. Your `applicationContext.xml` must look like the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/></property>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/></property>
        <property name="username" value="abc"/></property>
        <property name="password" value="xyz"/></property> </bean>
    <bean id="mysessionFactory" class="org.springframework.orm.hibernate3.
    LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/></property>
        <property name="mappingResources">
            <list>
                <value>student.hbm.xml</value> </list> </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.show_sql">true</prop>
            </props> </property> </bean>
        <bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
            <property name="sessionFactory" ref="mysessionFactory"/></property> </bean>
        <bean id="d" class="com.ith.StudentDao">
            <property name="template" ref="template"/></property> </bean>
    </beans>
```

You can then generate an Insert class which makes use of the StudentDao object and calls out the saveStudent method through the object passing of Student class.

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class Insert {
    public static void main(String[] args) {
        Resource res=new ClassPathResource("applicationContext.xml");
        BeanFactory fact=new XmlBeanFactory(res);
        EmployeeDao d=(EmployeeDao)fact.getBean("d");
        Student s=new Student();
        s.setId(102);
        s.setName("Mike");
        s.setMarks(77);
        d.saveStudent(s);
    }
}
```



How do we fetch data in case lazy loading is defined?

Summary

In the earlier days, software developers had to struggle a lot to link databases with Java applications. They had to consider an extensive list of factors and therefore were often left puzzled to find a solution. As a consequence, a lot of complex code was generated. Luckily, Hibernate's entry to the scene has changed things considerably.

Today, Hibernate has become an integral part of Java applications. Hence, if you plan to use databases heavily in your application, then do not take the hard way. Simply begin learning Hibernate and easily manage your objects and tables with utmost ease.

In this chapter, we have learned the following concepts:

1. How ORM tools work.
2. Use of hibernate in mapping classes with database tables.
3. How inheritance mapping works.
4. Various annotations to map entity relations.
5. How transactions work with Hibernate.
6. How caching works with Hibernate.
7. How to integrate with Spring.
8. How to fetch record with Hibernate Query Language.
9. How to use named query.

In Chapter 22, we will work on the project we have defined in Chapter 2. We have worked on the front end side of it and now we will work on setting up Spring MVC web services.

Multiple-Choice Questions

1. Which of the following objects is used to generate SessionFactory object in Hibernate?

(a) Session	(b) SessionFactory (c) Transaction (d) Configuration
-------------	--

2. Which of the following is not a fetching strategy?
 - (a) Sub-select fetching
 - (b) Select fetching
 - (c) Join fetching
 - (d) Dselect fetching
 3. What is the full form of QBC?
 - (a) Query By Column
 - (b) Query By Code
 - (c) Query By Criteria
 - (d) Query By Call
 4. Which of the following is the file where database table configuration is stored?
- (a) .ora
 - (b) .sql
 - (c) .hbm
 - (d) .dbm
5. Which one of the following is not a valid type of cache?
 - (a) First Level
 - (b) Second Level
 - (c) Third Level
 - (d) None of the above

Review Questions

1. What is ORM? What are the benefits of using it?
2. How can Hibernate connect with Spring MVC?
3. How do you define one-to-one relationship?
4. How do you define one-to-many relationship?
5. What are the essential annotations for setting up Hibernate entity on the Spring side?
6. How do you connect database with Hibernate?

Exercises

1. Design a database for managing inventory in a grocery store. Create all the tables and fields for the same and setup hibernate project for the same. Take examples from this chapter.
2. For the above project, create a list of primary and foreign keys that you will map with the variables from the Java class side.

Project Idea

For the Hotel Booking application we have created using Spring MVC in Chapter 20, create a database. Design tables and their relationships based on the model classes and think of the relationship mapping with annotations.

Recommended Readings

1. Christian Bauer, Gavin King, Gary Gregory, and Linda Demichiel. 2017. *Java Persistence with Hibernate, Second Edition*. Manning Publications: New York
2. Joseph B. Ottinger, Jeff Linwood, and Dave Minter. 2016. *Beginning Hibernate: For Hibernate 5*. Apress: New York
3. Yogesh Prajapati and Vishal Ranapariya. 2015. *Java Hibernate Cookbook*. Packt: Birmingham

Develop Web Services for the APIs

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- How to create a Spring MVC project using Spring Boot.
- How to create the REST web service endpoints.
- Practical use of MVC framework.
- Basic use of Eclipse based IDE like Spring Tool Suite.
- Data Access Object design pattern.
- Useful annotations like @Autowired, @RequestMapping, and @PathVariable.

22.1 | Setting up Environment



The foremost requirement for setting up the development environment is installing the Java Development Kit (JDK). For this project, we will use JDK 11. So please visit the official Oracle website to download the latest JDK 11 for your operating system. Once you have installed the JDK 11, you may begin installing the Integrated Development Environment (IDE).

Eclipse is the widely used open source IDE for Java development. Spring has extended eclipse IDE and created its own tool known as Spring Tool Suite (STS). So far, you have worked on Eclipse and must be familiar with its interface, now let us also try STS as Spring supports this better. At the time of writing, STS 4 was available for download. For this learning exercise, we will use STS where the experience is very similar to eclipse IDE.

Please download and install the required STS IDE from the URL <https://spring.io/tools>.

On the above page you will see the screen shown in Figure 22.1. Select the build based on your operating system. In our case, we have downloaded the Mac version.

QUICK CHALLENGE

List the differences between eclipse and Spring Tool Suite.

Once the download is done, find the build and install it on your system. This installation will be similar to other applications you install on your computer. Once the installation is done, you will find the icon of the STS build, which you can use to open the STS IDE (Figure 22.2).

Once you open the IDE, you will see the interface shown in Figure 22.3.

If you are new to eclipse environment, we recommend you go through the official documentation to get familiarise yourself with the environment. You can find some useful information on the official page https://www.eclipse.org/getting_started.

Once you are familiar with the interface, you can start creating a Spring MVC based REST web service using Spring Boot.

The screenshot shows the Spring Tool Suite 4 interface running on Eclipse. The workspace contains a Java project named "greet-service-complete". The "GreetingController.java" file is open, showing code for a REST controller that returns a greeting message. Other files visible include "MyAutowiredComponent.java", "MyConfiguration.java", "MyService.java", "MySpecialService.java", and "MyUtil.java". The interface includes standard Eclipse toolbars and menus like "File", "Edit", "Search", "Run", and "Help". A "Properties" view is open on the right side.

Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.
Free. Open source.

Download STS4
Linux 64-bit

Download STS4
macOS 64-bit

Download STS4
Windows 64-bit

Figure 22.1 Spring Tool Suite 4 download page.



Figure 22.2 Spring Tool Suite 4 icon.

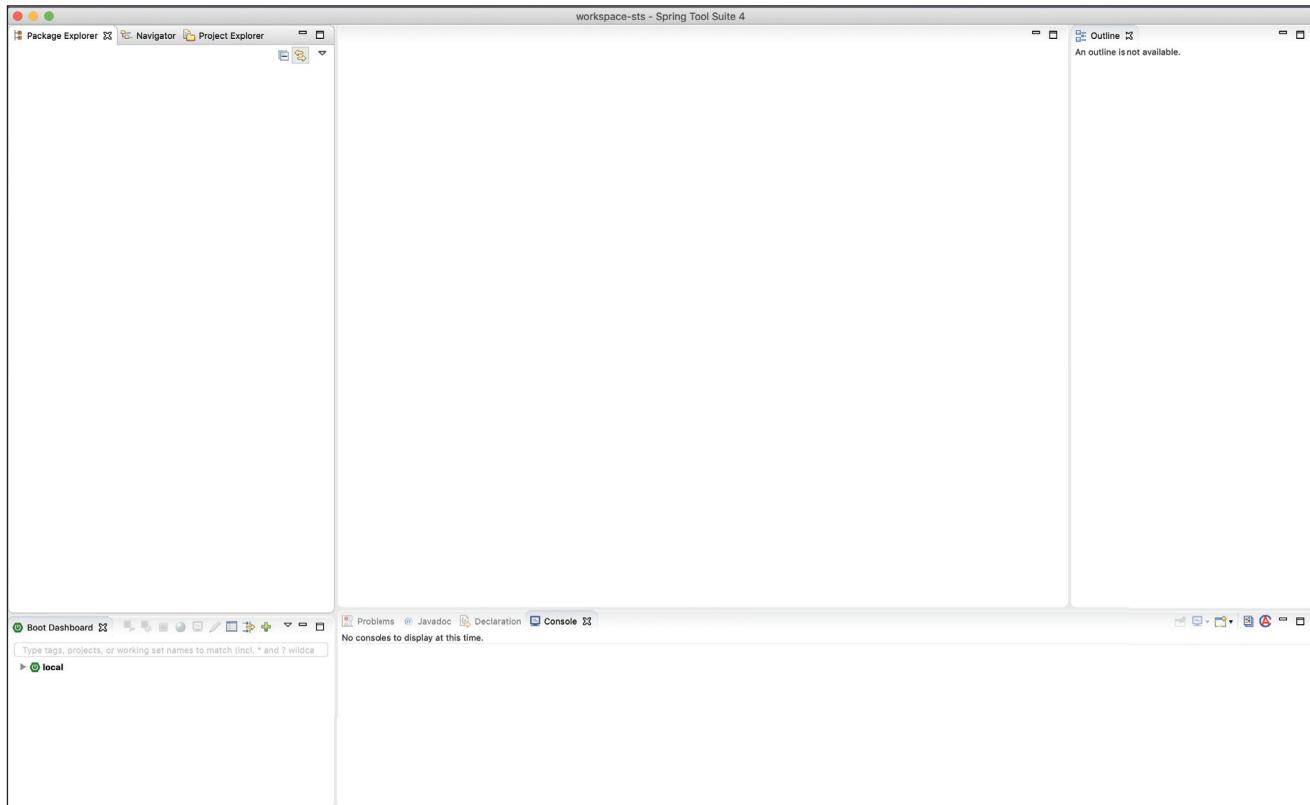


Figure 22.3 Spring Tool Suite 4 IDE.



What is Spring Boot? What is the use of it?



22.2 | Creating a New Project

The first step is to click on the File menu. Then click on New->Spring Starter Project. See Figure 22.4 to find the location of this menu.

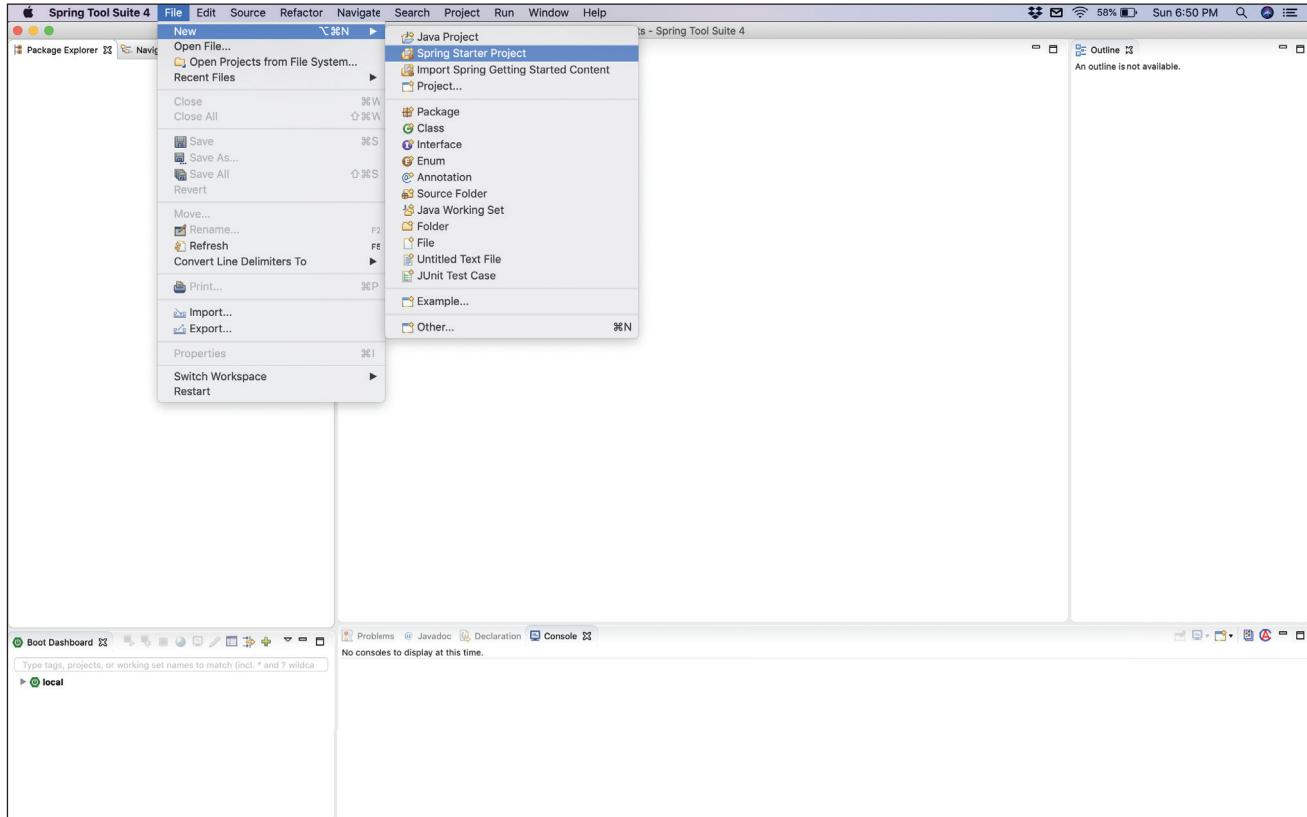


Figure 22.4 Menu to create a new project.

Once you clicked on the menu, you will see the screen shown in Figure 22.5, where you will enter the project related information as follows:

1. **Service URL:** This field gives us the standard URL which we can keep it as it is.
2. **Name:** This field is to specify the project name. In our case, we will name our project as MyEShop.
3. **Location:** This file is to specify project location where the project will be saved.
4. **Type:** In our case, we will be selecting Maven. Maven is a build automation tool which greatly simplifies the build process. We will see how easy it is to manage the dependencies with Maven.
5. **Java version:** In this field, we will specify the Java version which we will use in the development. In our case, we will select Java 11.
6. **Packaging:** This field is for selecting the packaging type we need for our project. We are developing web services that will be consumed by our front end; hence we can select War as a packaging type.
7. **Language:** This option is to specify our programming language. In our case, it is Java.
8. **Group:** This field is to specify the group for our project. We can specify any text we like.
9. **Artifact:** This will be the name of the project we will be creating.
10. **Version:** This field is to specify the project version. Since we are creating a new project, we can start with 0.0.1-SNAPSHOT version. We can start with any number we like.
11. **Description:** This field is to give the project description. It is a good practice to give good project description.
12. **Package:** This field is to specify the primary package for our project.
13. **Working set:** This section is to add our project to an existing working set. It is an optional step so we can skip it.

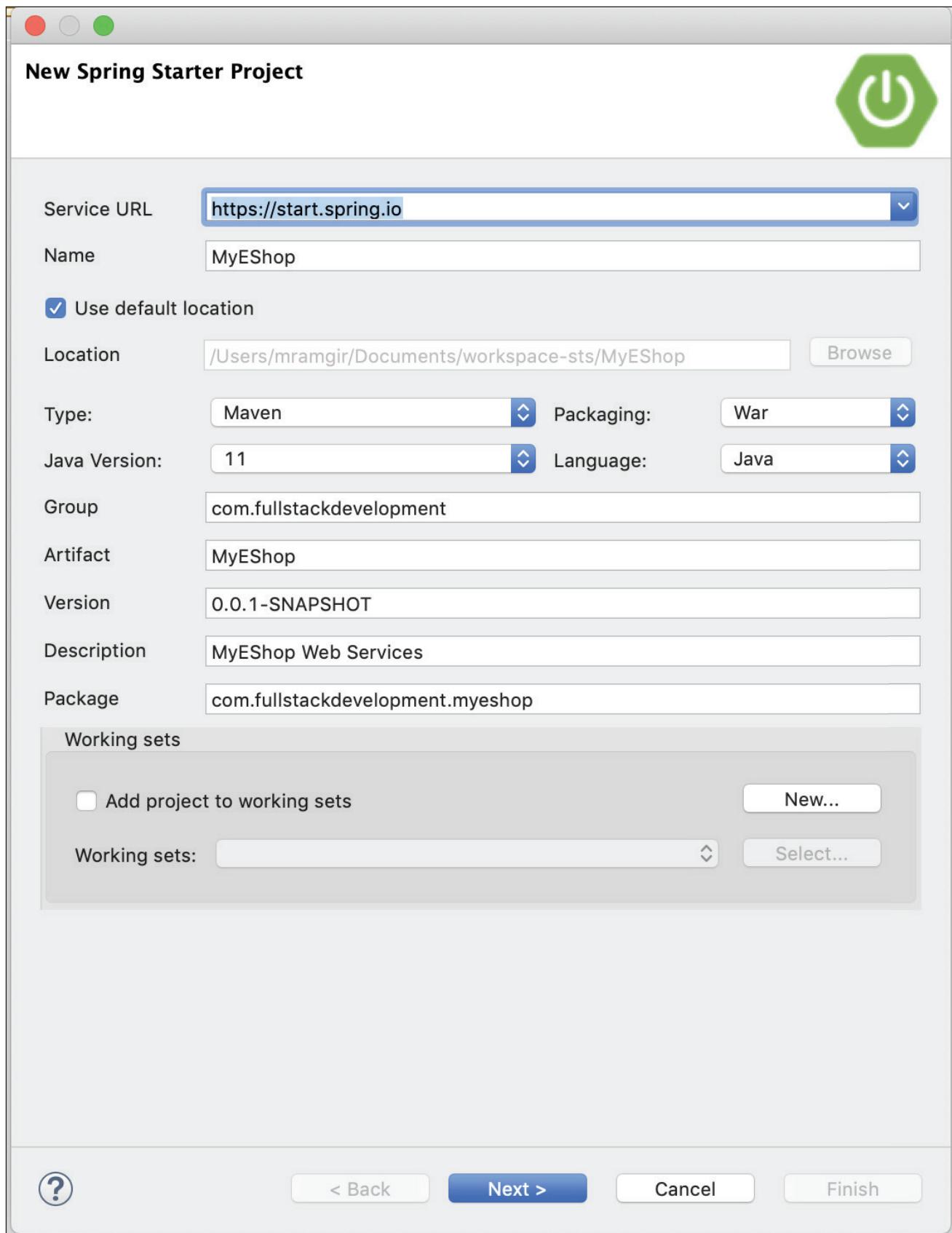


Figure 22.5 New starter project details window.

After you click on the Next button, you will see the next screen (Figure 22.6), which allows you to select project dependencies. You will see a lot of options that you can add to your project. In our case, we will be very selective in choosing, based on our requirement. In this screen, you will see an option called *Spring Boot Version*. Select the latest one you can see in the dropdown.



Figure 22.6 New starter project dependencies window.

For our needs, we will select web package which will allow us to use Spring MVC and a few SQL options, such as JPA and MySQL, that will allow us to connect with database using object-relational mapping (ORM) tools like Hibernate (Figures 22.7 and 22.8).

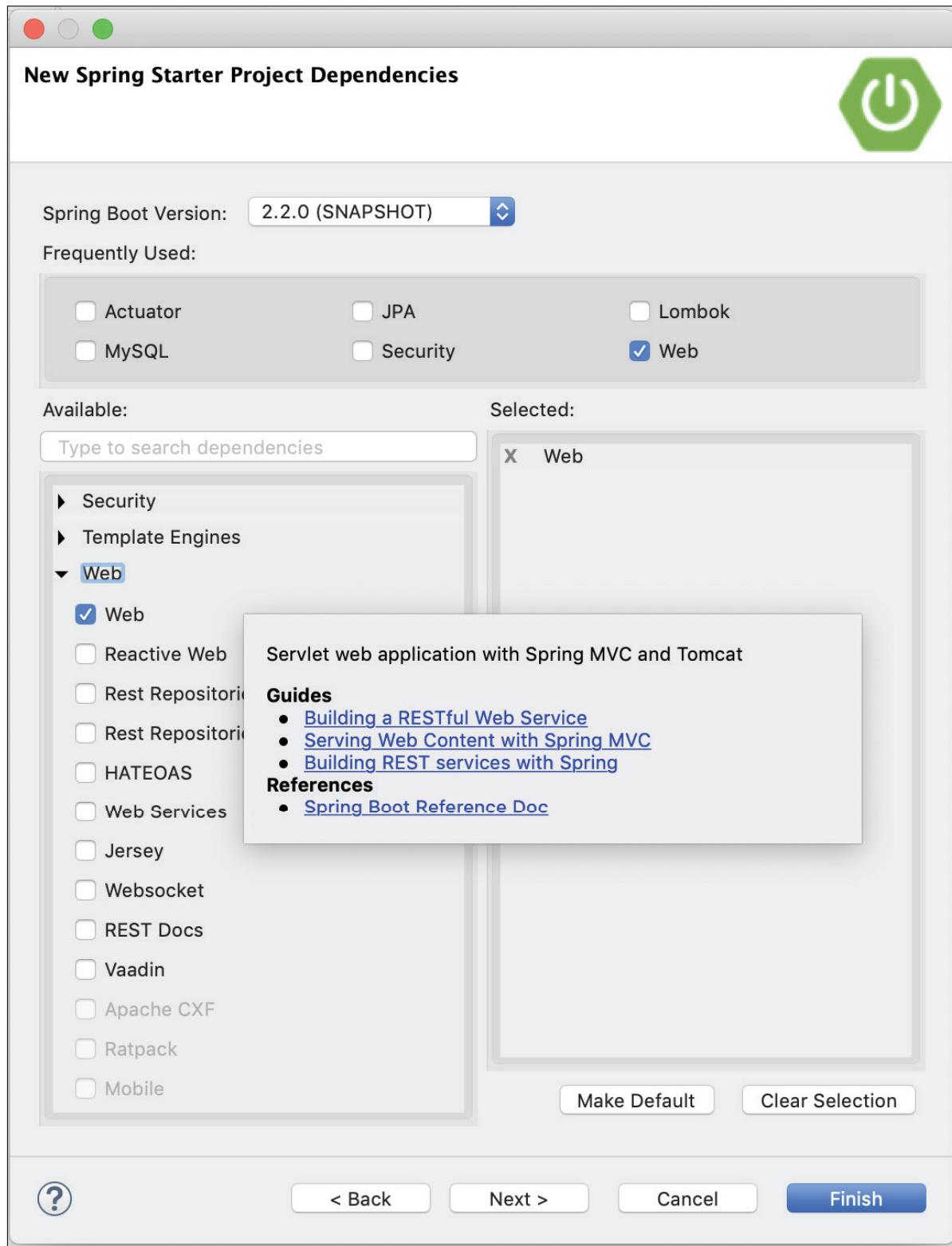


Figure 22.7 Select web package option.

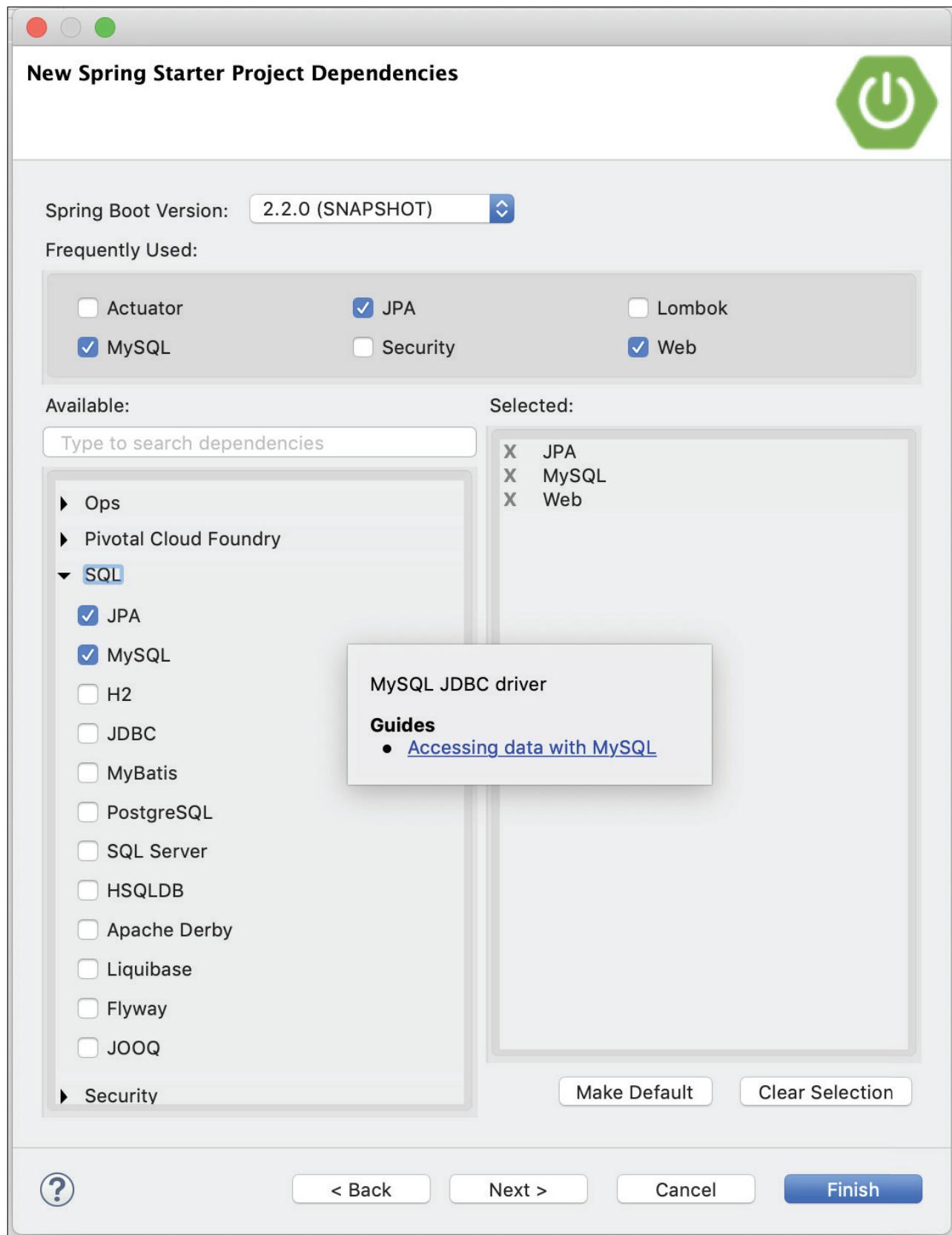


Figure 22.8 Select JPA and MySQL options.

After the desired selection, you may click on the **Next** button to continue. Upon clicking on the **Next** button, you will see the screen shown in Figure 22.9.

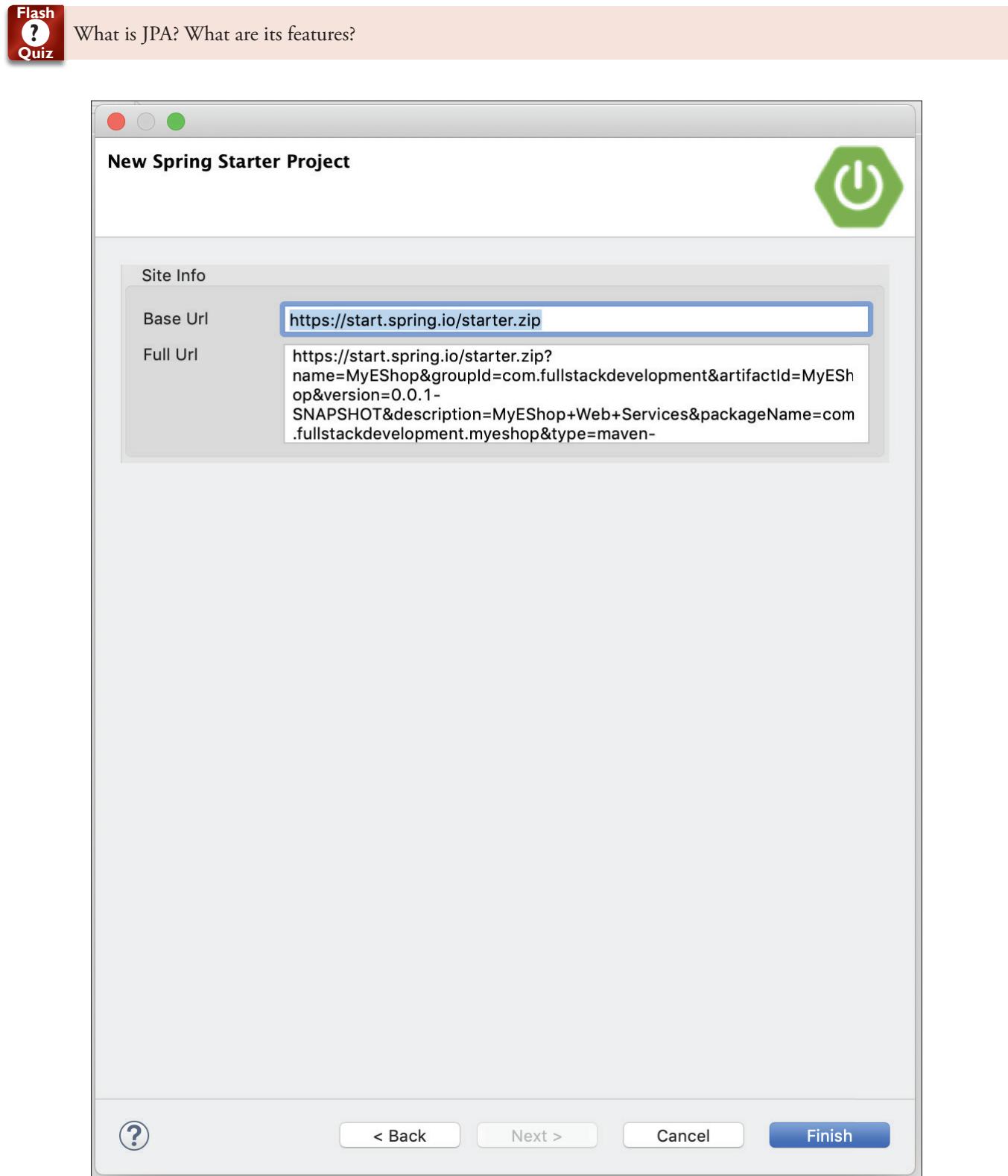


Figure 22.9 Window to show base and full URLs of the application.

There is nothing we need to do on this screen, as Spring Boot will create the project we need. So just click on the **Finish** button, which will create the project based on the values we have provided earlier.

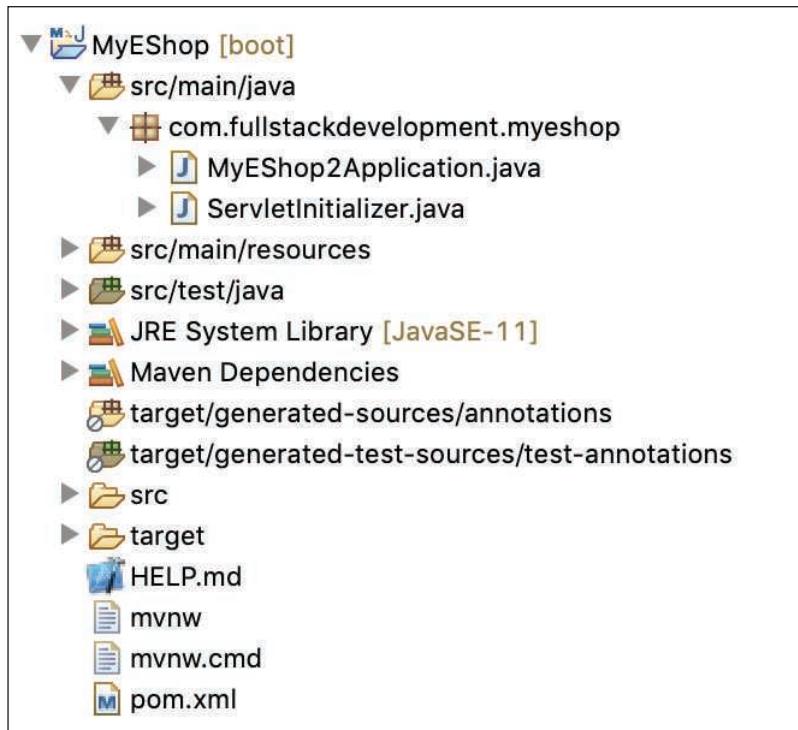


Figure 22.10 Explorer view of the project to see all the files.

Figure 22.10 shows the project structure. The blue icon shows the project name MyEShop. After that, we have initial files that Spring Boot has created for us under `src/main/java`. Under this folder we have our package folders like `com/fullstackdevelopment/myeshop`, which is shown in STS as `com.fullstackdevelopment.myeshop`. At the very end, you can see a file called `pom.xml`. This file is a Maven file, which contains the dependency settings.

Let us explore `pom.xml` file and learn about specifying dependencies for our project. The first section that is shown below is for specifying the project details.

```
<groupId>com.fullstackdevelopment</groupId>
<artifactId>MyEShop</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>MyEShop</name>
<description>MyEShop Web Services</description>
```

These are the details we have provided while creating the project through New->Spring Starter Project menu.

**QUICK
CHALLENGE**

Explain `pom.xml` in detail. List at least two of its benefits.

The next section given below is for specifying the project dependencies.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

In the above image, you can see three dependencies that we have selected while creating the project and the fourth one is added by Spring Boot. The following shows the repositories information, which are part of our project.

```
<repositories>
    <repository>
        <id>spring-snapshots</id>
        <name>Spring Snapshots</name>
        <url>https://repo.spring.io/snapshot</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
    </repository>
</repositories>
```

With this we are done setting up our project environment. Now, let us begin with creating required web services using Spring MVC.



What is dependency in pom.xml?

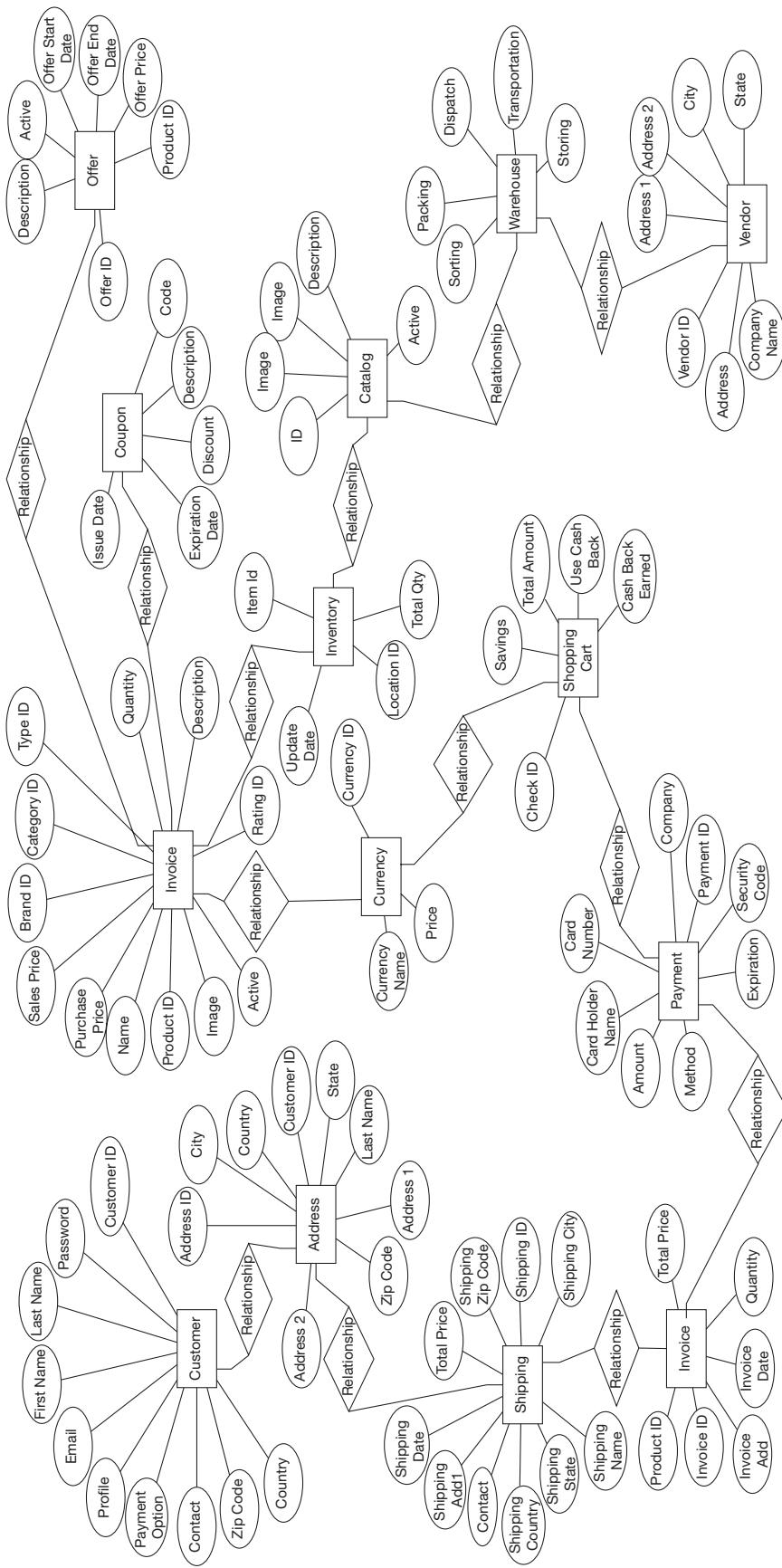


Figure 22.11 Entity relationship diagram of e-commerce entities.

22.3 | Creating Models



In this section, we will create models which are POJOs but are correlated with database tables by Hibernate. We will see this connection in Chapter 23, when we will explore Hibernate examples. But for now, let us look at these POJOs from database tables' point of view. We will use the Entity Relationship Diagram (ERD), which we have created earlier and that you have learned in Chapter 2. This ERD will give us an idea about the models we need to create. These are the entities defined in the ERD. Let us see the diagram again to refresh your memory (Figure 22.11).



Which of the two is considered as an entity – persistence classes as entities or non-persistence classes? Explain

Now, let us create a package for Models. Right click on the project name and click on New->Package as shown in Figure 22.12.

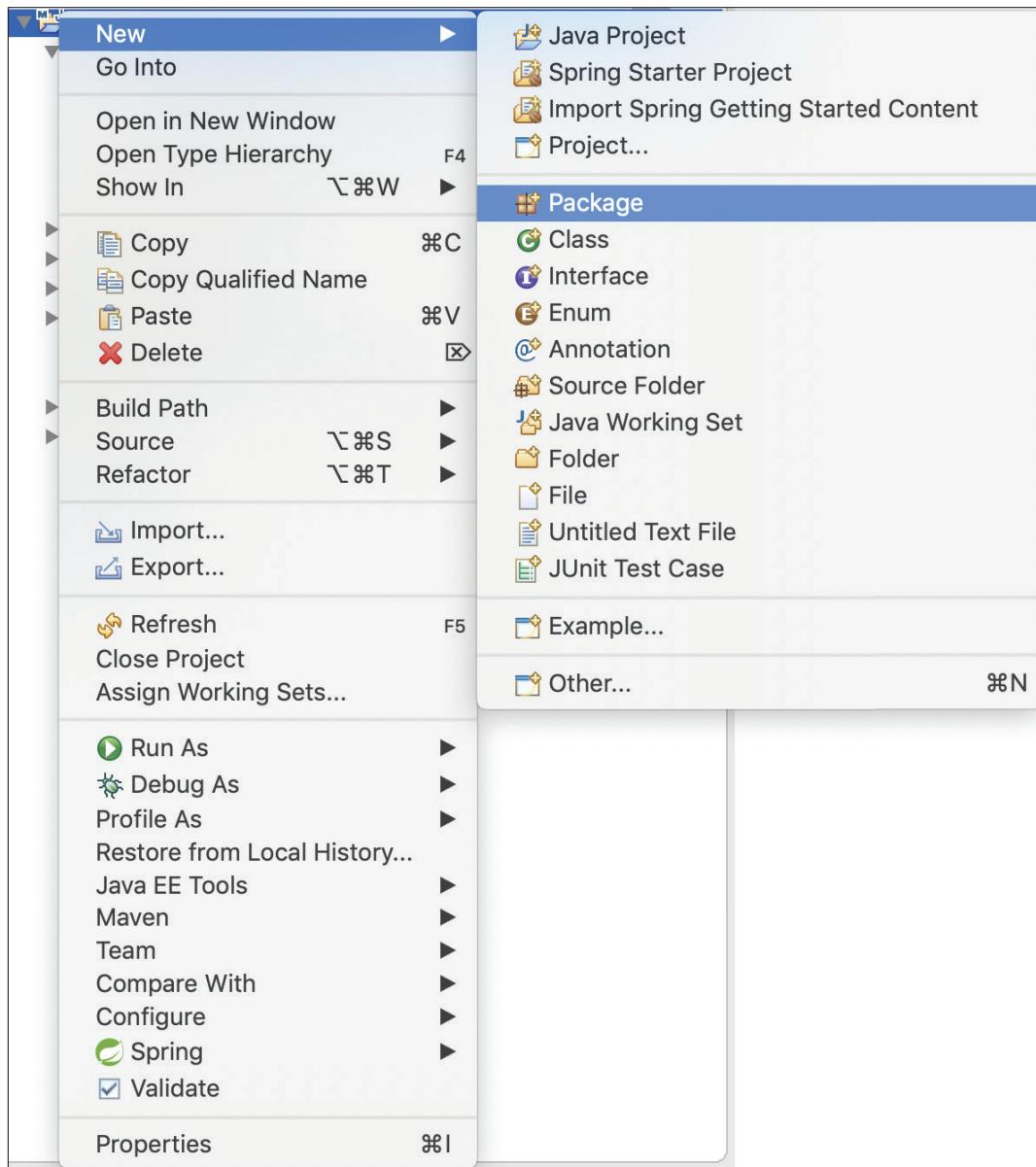


Figure 22.12 Menu to create a package.

After clicking on the menu, you will see the screen shown in Figure 12.13, where you can specify the name of the package.

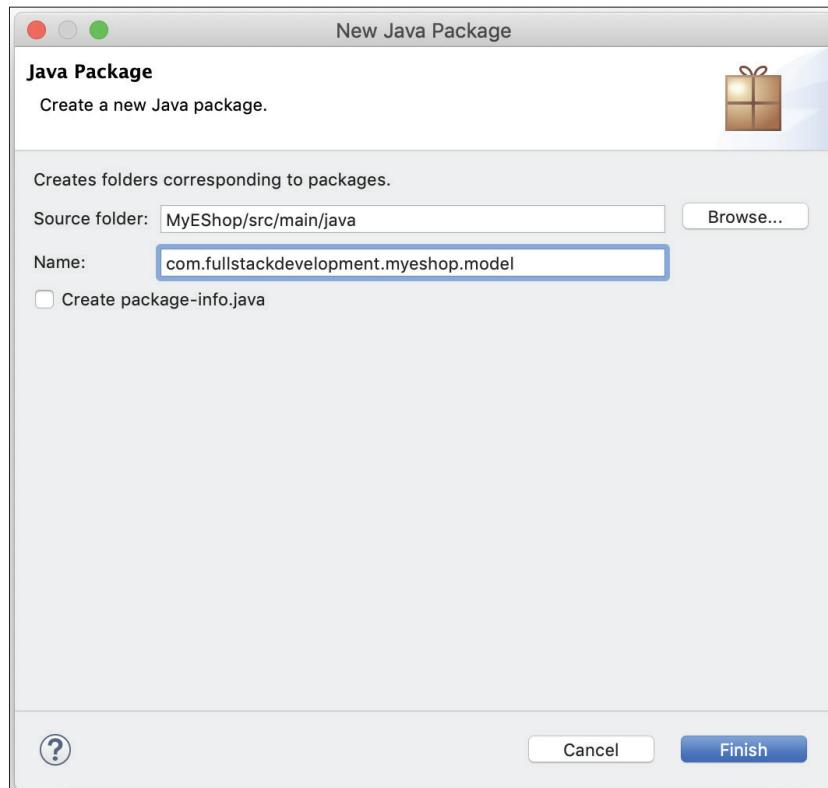


Figure 22.13 Create Java package window.

Once you click on the **Finish** button, you will see the newly created package in the package explorer as shown in Figure 22.14.

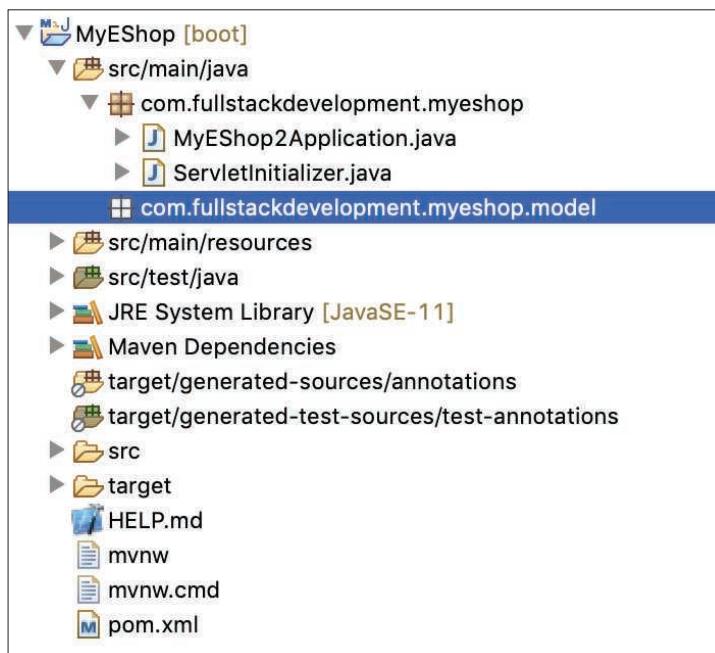


Figure 22.14 Newly created Java package.

Now, let us add model POJOs that we need. For this, right click on the newly created package `com.fullstackdevelopment.myeshop.model` and click on "Class" as shown in Figure 22.15.

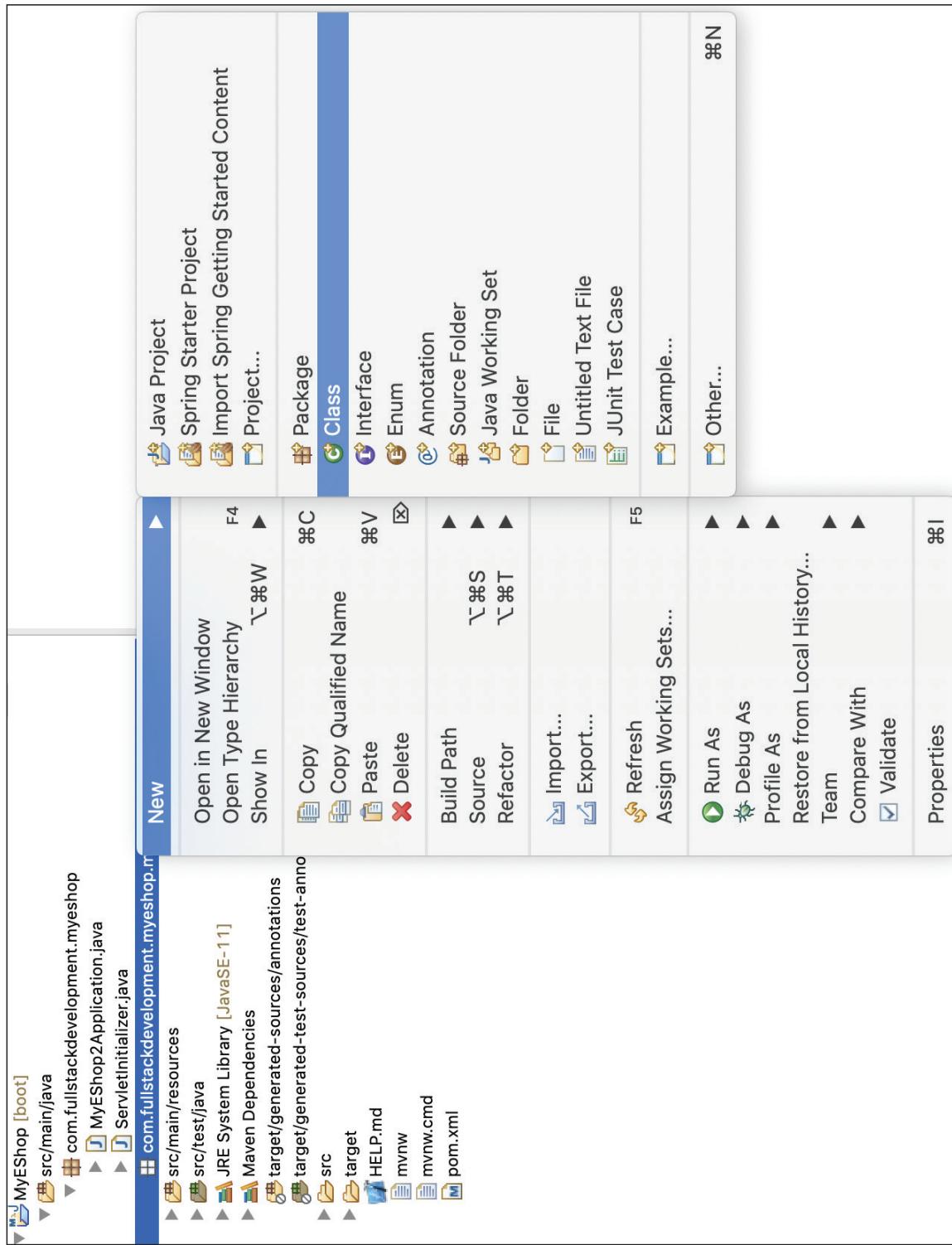


Figure 22.15 Menu to create a new class.

Once you click on the menu, you will see a window in which you can specify the class name as shown in Figure 22.16. In this case, we will be creating Customer model.

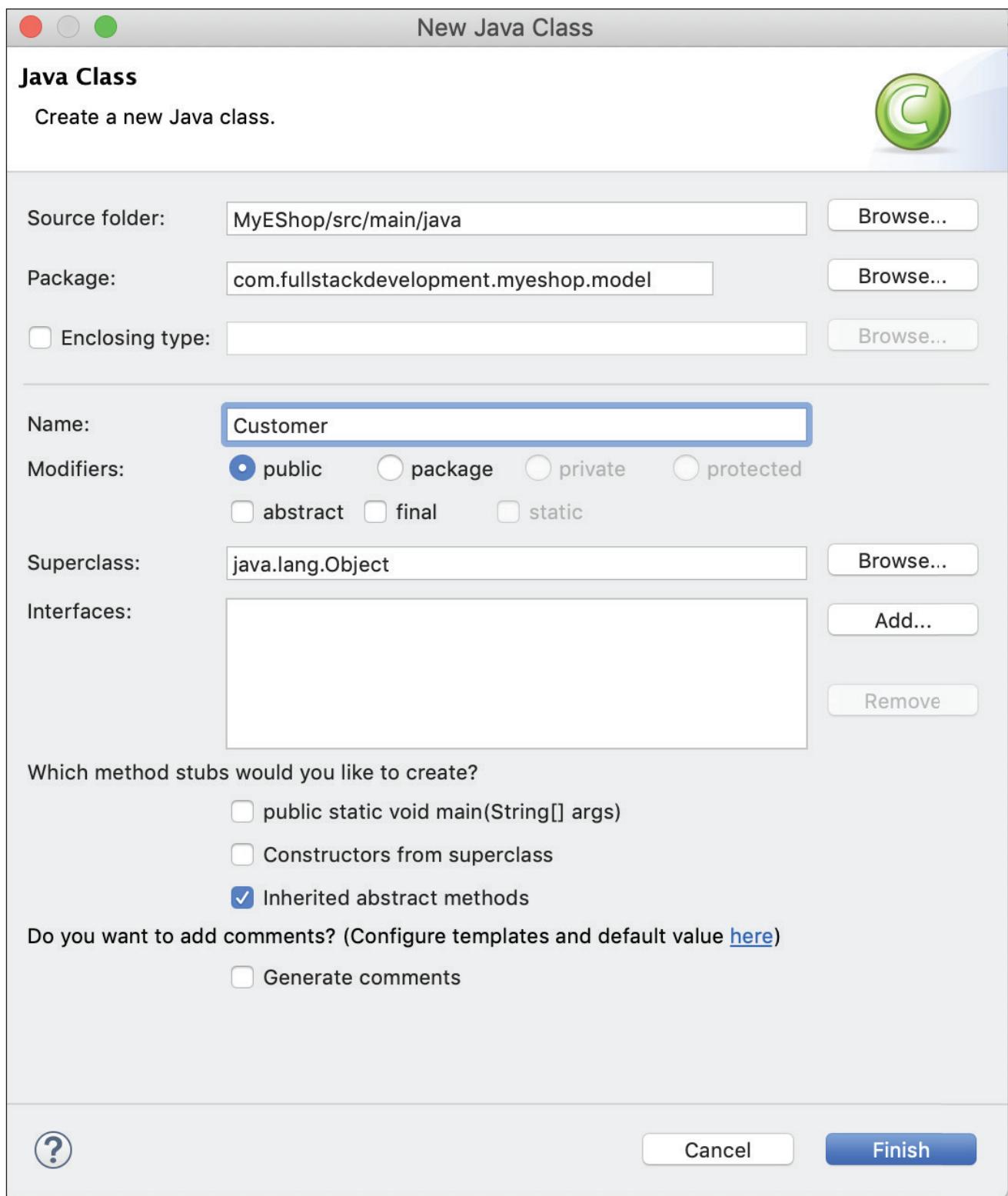


Figure 22.16 Java class details window.

Once you click on the Finish button, you will see the newly created Customer class in the package explorer under package com.fullstackdevelopment.myeshop.model. See Figure 22.17.

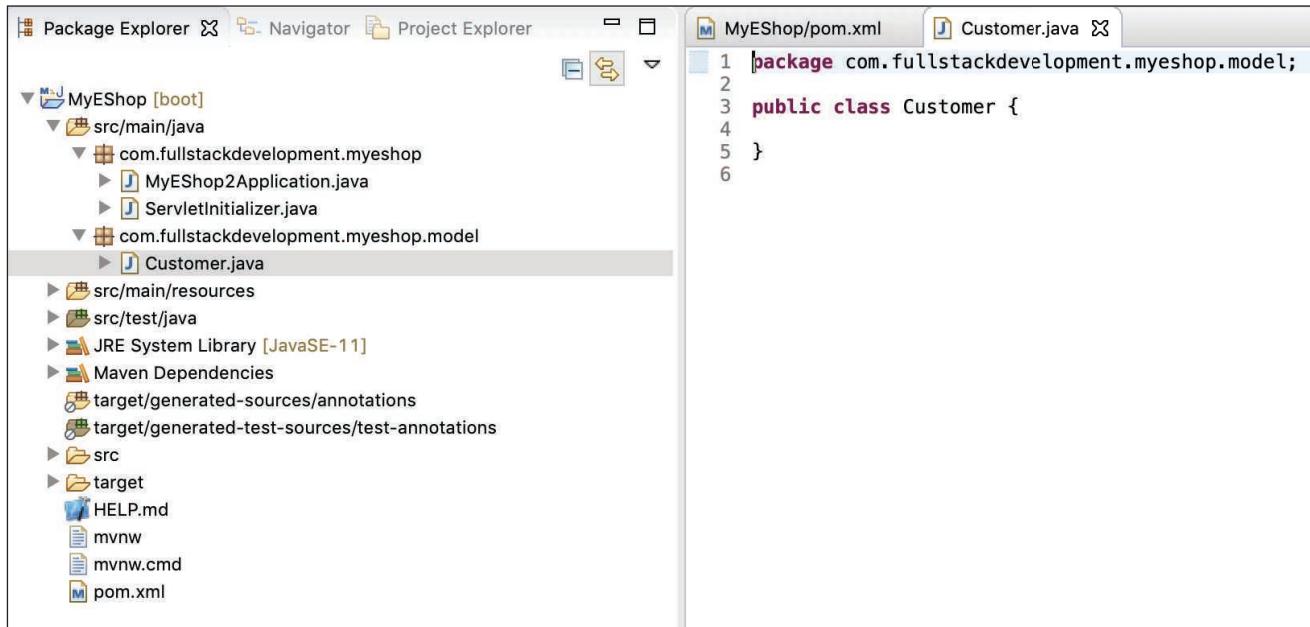


Figure 22.17 Newly created class.

Now we need to define this Customer class as Entity in order to use it with Hibernate. It is as simple as adding @Entity annotation above the Customer class definition as shown in Figure 22.18.

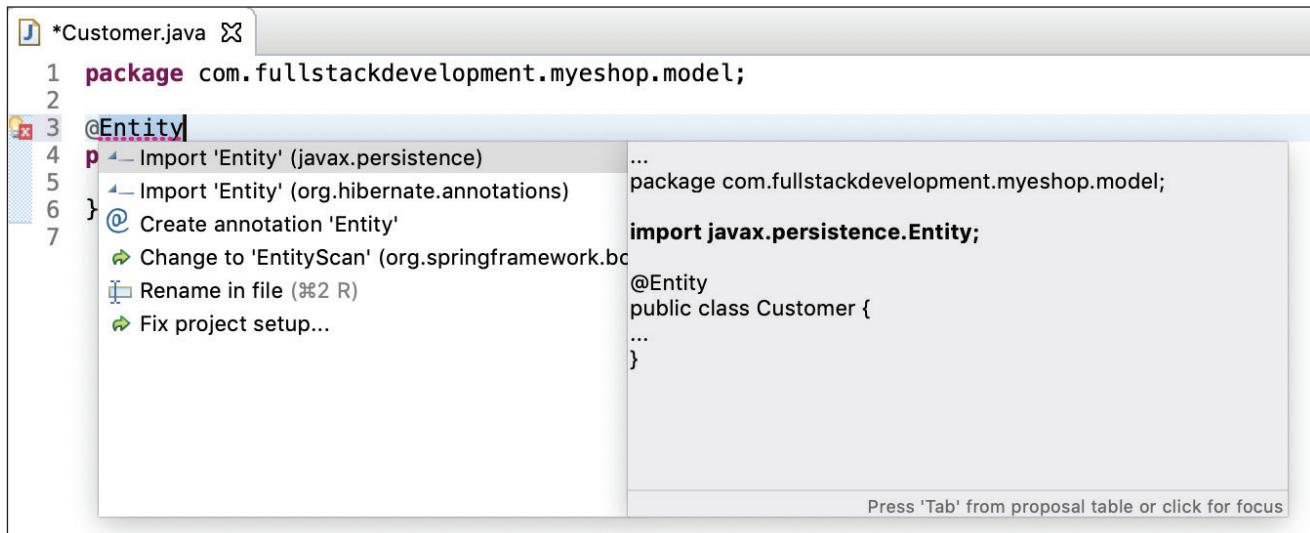


Figure 22.18 Auto-suggest dropdown for import.

We need to import the Entity annotation in order to use it. As you can see in Figure 22.18, there are two options – one from javax.persistence and other from org.hibernate.annotations. Although we will be using Hibernate, we do not want to make this very specific to Hibernate and hence we will be selecting Entity from javax.persistence package.

```

J *Customer.java ✘
1 package com.fullstackdevelopment.myeshop.model;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class Customer {
7
8 }
9

```

We need to tell Spring the table name this Entity is going to be associate with. For this, we need to add `@Table` annotation from `javax.persistence`.

```

J *Customer.java ✘
1 package com.fullstackdevelopment.myeshop.model;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 @Table
7 p Import 'Table' (javax.persistence)
8   ↗ Import 'Table' (org.hibernate.annotations)
9 } @ Create annotation 'Table'
10   ↗ Change to 'Tables' (org.hibernate.annotations)
    ↗ Rename in file (⌘2 R)
    ↗ Fix project setup...
... import javax.persistence.Entity;
import javax.persistence.Table;
...

```

Press 'Tab' from proposal table or click for focus

Once the import is done, we need to specify the name attribute. In our case, we will name the table as “customer”.

```

J Customer.java ✘
1 package com.fullstackdevelopment.myeshop.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Table;
5
6 @Entity
7 @Table(name="customer")
8 public class Customer {
9
10 }
11

```

Now, it is time to add attributes to the Customer model. These attributes can be customer name, customer address, customer date of birth, etc. To start, we will add simple fields and later we can see the improved version of this entity with other entities related to it. For example, for now we have used address as a String field. However, address can be a separate model that can store multiple addresses such as shipping address, billing address, etc. So, this address entity will be related to Customer entity in “many-to-one” fashion and from Customer point of view in “one-to-many” fashion. In Chapter 23, we will explore this in detail. For now, let us continue with a simple example.

QUICK CHALLENGE

Define @Entity and @Table annotations in detail.

```

J Customer.java ✎
1 package com.fullstackdevelopment.myeshop.model;
2
3 import java.sql.Timestamp;
4 import javax.persistence.Entity;
5 import javax.persistence.Table;
6
7 @Entity
8 @Table(name="customer")
9 public class Customer {
10
11     private String name;
12     private String address;
13     private Timestamp dateOfBirth;
14
15     public String getName() {
16         return name;
17     }
18     public void setName(String name) {
19         this.name = name;
20     }
21     public String getAddress() {
22         return address;
23     }
24     public void setAddress(String address) {
25         this.address = address;
26     }
27     public Timestamp getDateOfBirth() {
28         return dateOfBirth;
29     }
30     public void setDateOfBirth(Timestamp dateOfBirth) {
31         this.dateOfBirth = dateOfBirth;
32     }
33
34 }
```

As you can see in the above example, we have added getter and setter for each field. This is the most important step we need to perform as Spring is going to use these fields to inject values to this Entity object. Hence, make sure you add getter and setter

for all the fields you add. The above example also shows a new field called Timestamp, which is imported from java.sql. As we will be storing this in the database, we need to make sure it is compatible with that.

QUICK CHALLENGE

Using the examples shown in Section 22.3 of Customer, create all other models which we have shown in the ERD. Start with basic fields.

Now we will move on to creating Data Access Object (DAO) for this Customer model. This layer is important in terms of keeping database interaction related code. DAO will access models to fetch the required data.

22.4 | Creating Data Access Object



In this part, we will create DAO for the Customer model we have created in Section 22.3. DAO encapsulates the details of models (persistence layer) and provides an interface for database operations for adding, retrieving, updating, and deleting data. This mechanism supports the single responsibility principle.

Now, let us create a package for DAO. Use the same steps to create a new package as we did earlier for model. Right click on the project and click on New->Package as shown in Figure 22.19.

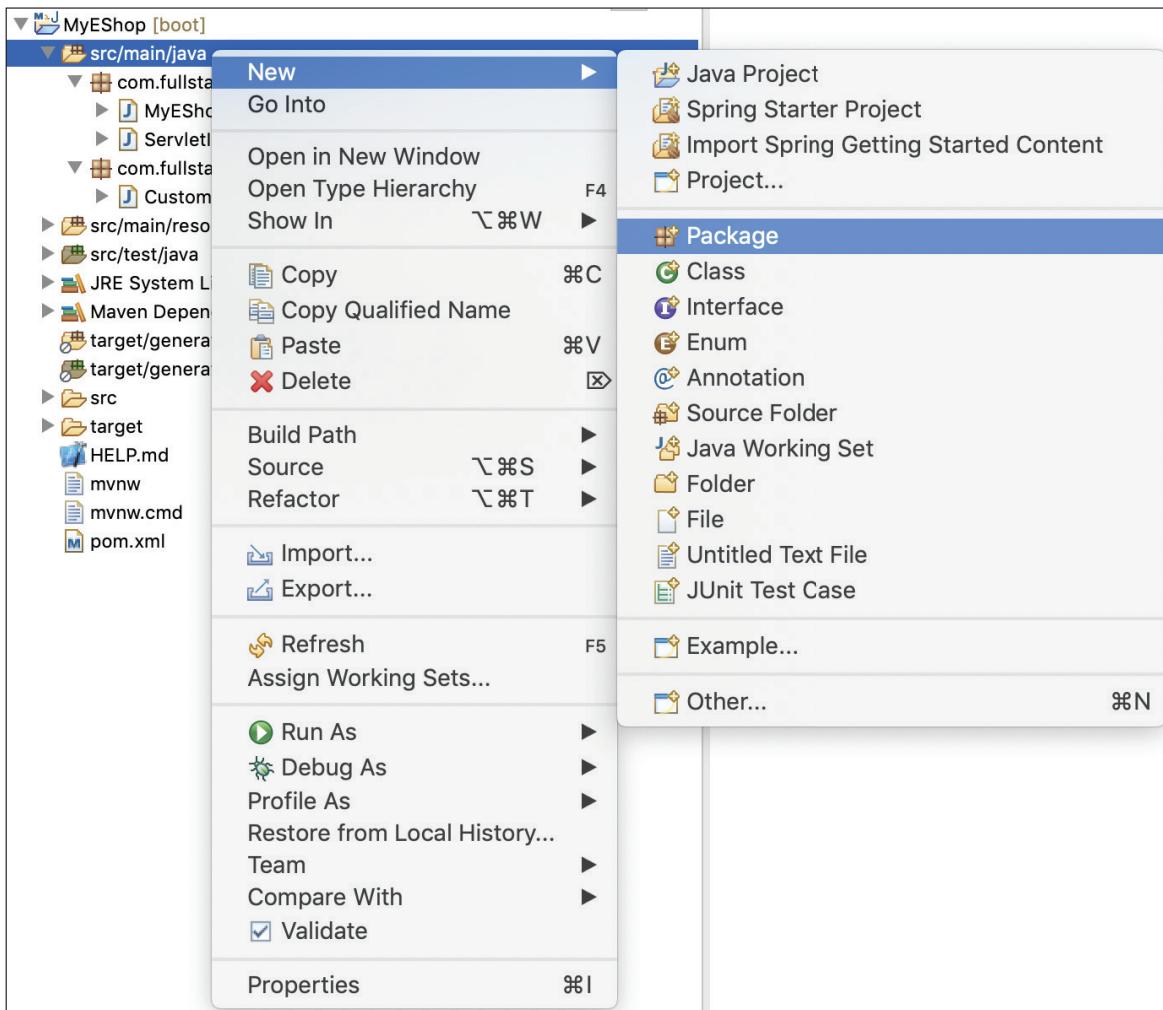


Figure 22.19 Menu to create a new package.

Once you click on the package menu, you will see the screen to enter the details about this new package as shown in Figure 22.20.

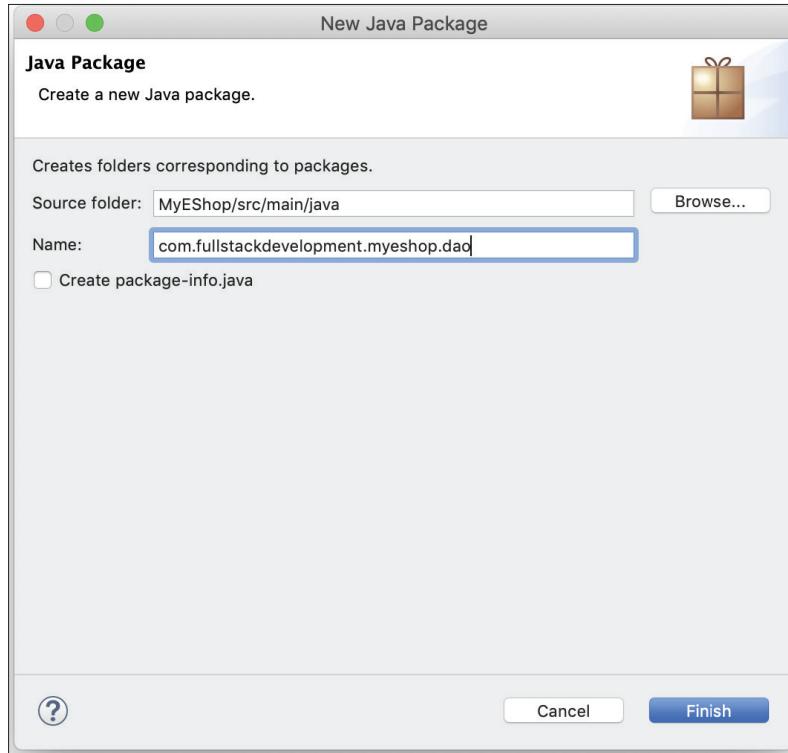


Figure 22.20 A new package detail window.

After entering the name and clicking on the Finish button, it will show you the newly created package in the package explorer (Figure 22.21).

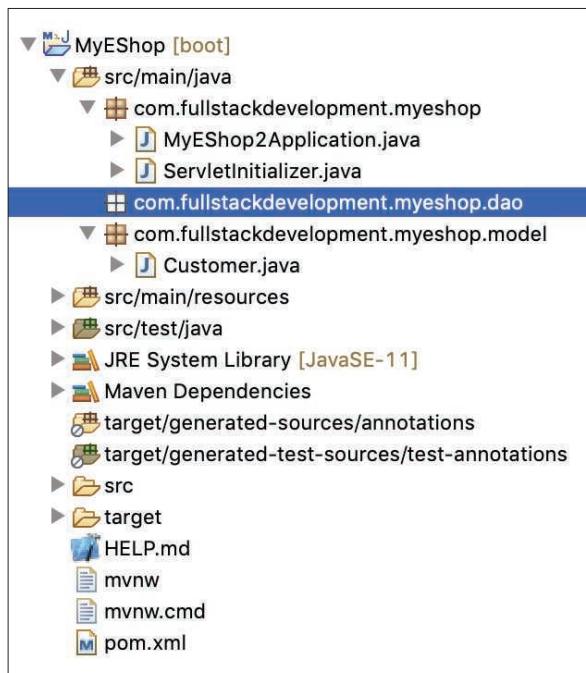


Figure 22.21 A newly created package.

In this package, we can add our DAO. Before adding any specific DAO classes, we can create an interface which all the DAO interfaces can extend that provides operations which will be useful for all the classes like get, getAll, save, update, delete, etc. Let us create this GenericDAO interface. For this, right click on the package and click on New->Interface as shown in Figure 22.22.

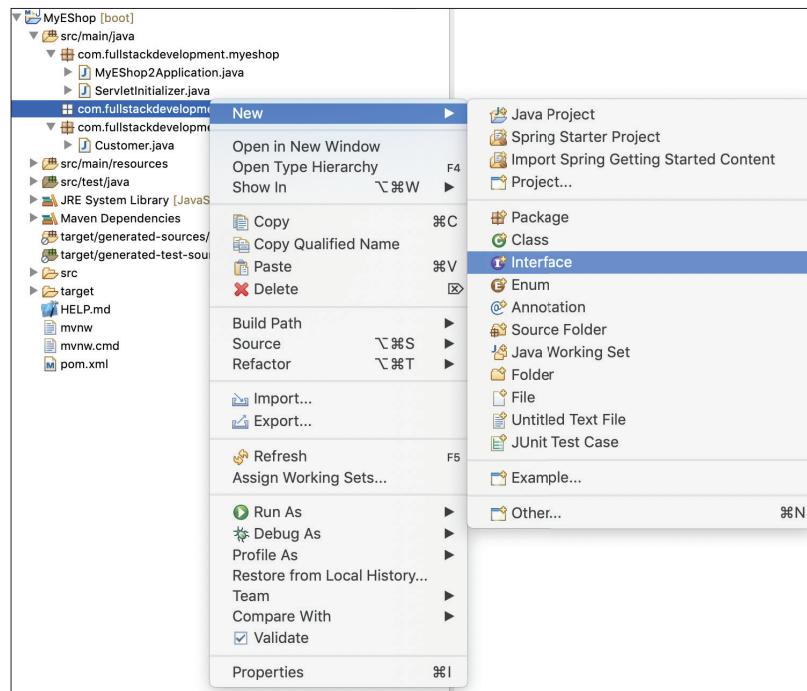


Figure 22.22 Menu to create a new interface.

Upon clicking on New->Interface, you will see a window which will allow you to enter the interface related information. See Figure 22.23.

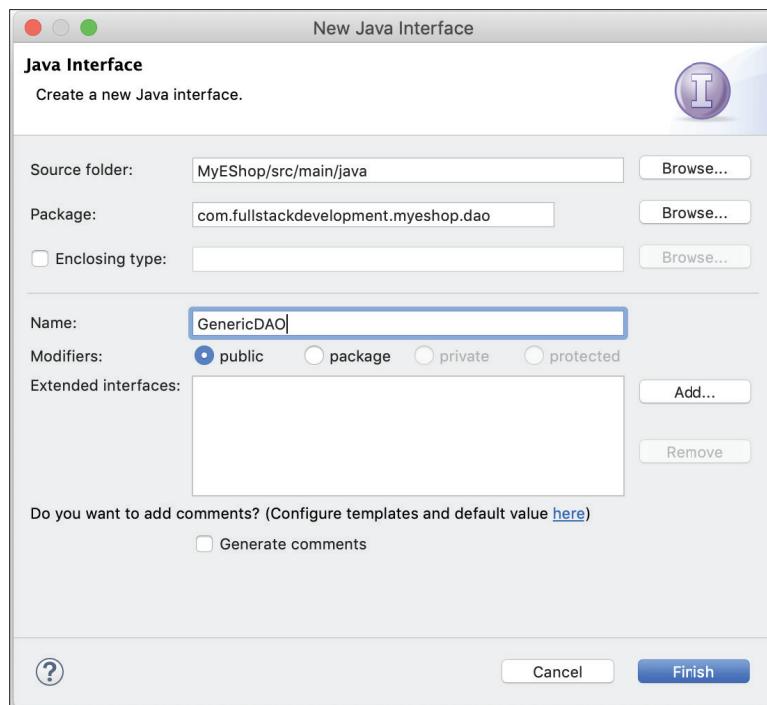


Figure 22.23 New interface detail window.

Enter DAO interface name as “GenericDAO” and click on the Finish button. As shown in Figure 22.24, this will create the interface we need.

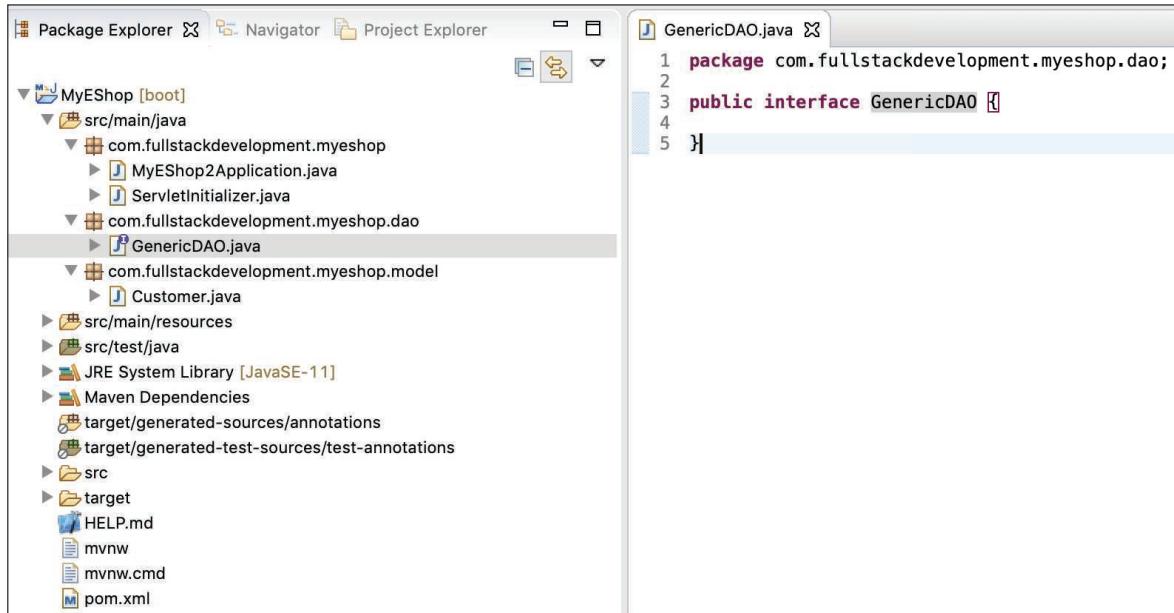


Figure 22.24 Newly created interface.

We need to add type for this interface so the model we will be implementing in can replace it.

```

1 package com.fullstackdevelopment.myeshop.dao;
2
3 public interface GenericDAO<T> {
4
5 }

```

Now, we can add the method declarations that will be useful to all the models. It is time to import the class, take the mouse pointer to the highlighted word and click on the import package. For List, we will use java.util.

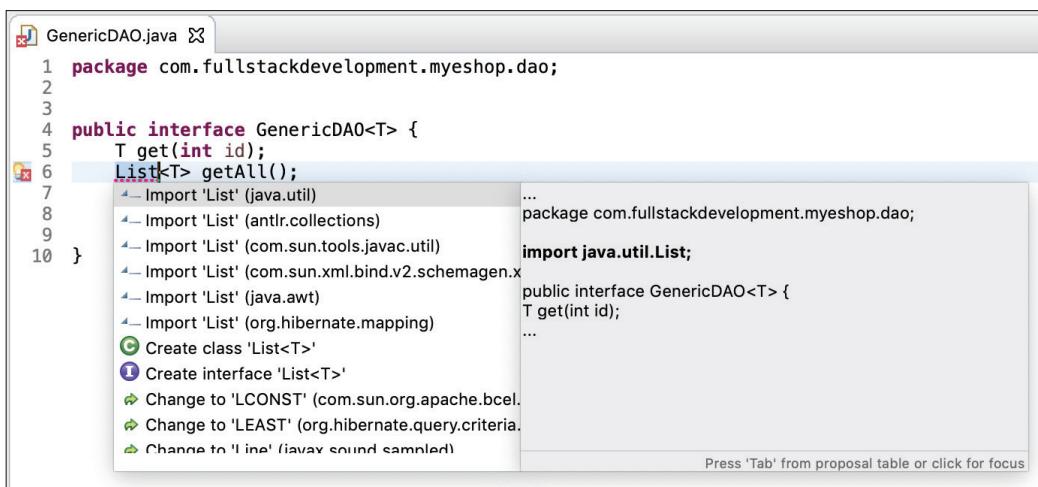


Figure 22.25 Auto-suggest dropdown to import List class.

As shown in Figure 22.25, we are importing the List class from java.util package. Once the import is done, your code should look as shown below.

```

1 package com.fullstackdevelopment.myeshop.dao;
2
3 import java.util.List;
4
5 public interface GenericDAO<T> {
6     T get(int id);
7     List<T> getAll();
8     Long save(T t);
9     void update(T t);
10    void delete(T t);
11 }

```

We can extend this GenericDAO to all the DAO interfaces we will be creating. This will avoid the code duplication and we can still add model specific interface methods. Let us try and understand this better with the help of an example. So let us create a CustomerDAO interface to have Customer specific methods (Figure 22.26). Let us use the same step to create an interface and give name it CustomerDAO.

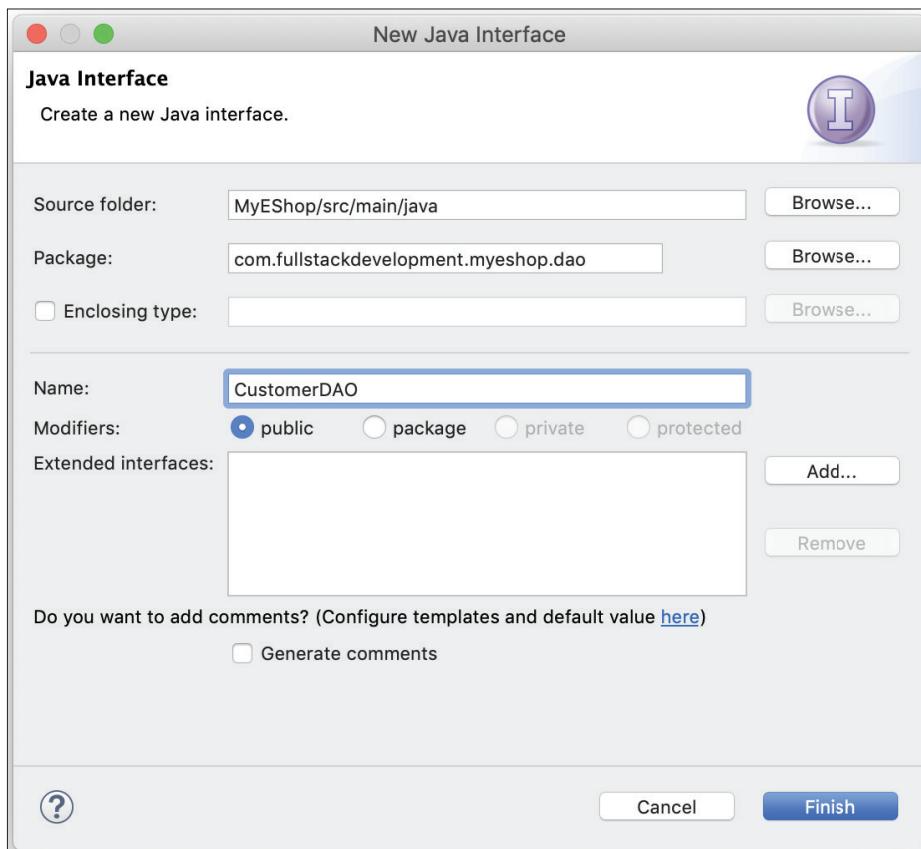


Figure 22.26 New interface detail window.

Once you click on the Finish button, you will see the newly created interface in the project explorer under com.fullstackdevelopment.myeshop.dao package. Now, let us extend this interface from GenericDAO and use type as Customer. See the below example.

```

1 package com.fullstackdevelopment.myeshop.dao;
2
3 import com.fullstackdevelopment.myeshop.model.Customer;
4
5 public interface CustomerDAO extends GenericDAO<Customer>{
6
7 }
8

```

This is a great example of object-oriented programming (OOP) principle. In this example, we are accessing all the method declarations from GenericDAO in CustomerDAO interface and making them available to all the classes, which will implement CustomerDAO interface. In CustomerDAO interface, we are free to declare any Customer specific methods which will only be available to classes which implements CustomerDAO. See the following example.

```

1 package com.fullstackdevelopment.myeshop.dao;
2
3 import com.fullstackdevelopment.myeshop.model.Customer;
4
5 public interface CustomerDAO extends GenericDAO<Customer>{
6     Customer findCustomerByEmail(String email);
7 }
8

```

QUICK CHALLENGE

As per the examples shown in 22.4, create DAO interfaces for all the models you have created in the earlier exercise.

Now, it is time to create a concrete DAO class that will implement this CustomerDAO interface. Right click on the project and add a new package as we have added before and give name it com.fullstackdevelopment.myeshop.dao.impl as shown in Figure 22.27.

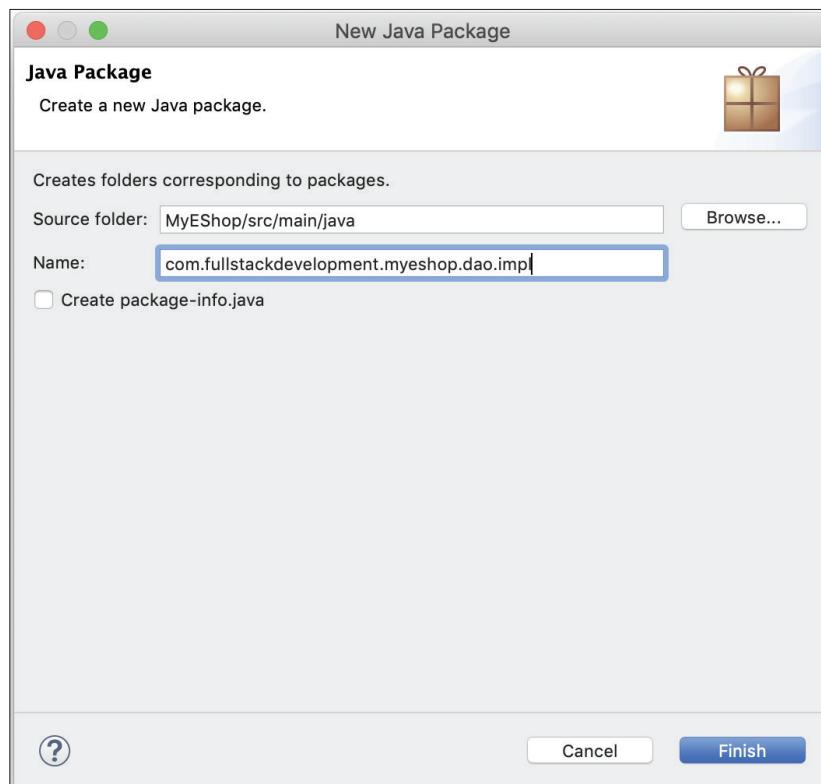


Figure 22.27 New package detail window.

Once you click on the Finish button, you will see “com.fullstackdevelopment.myeshop.dao.impl” in the project explorer as shown in Figure 22.28.

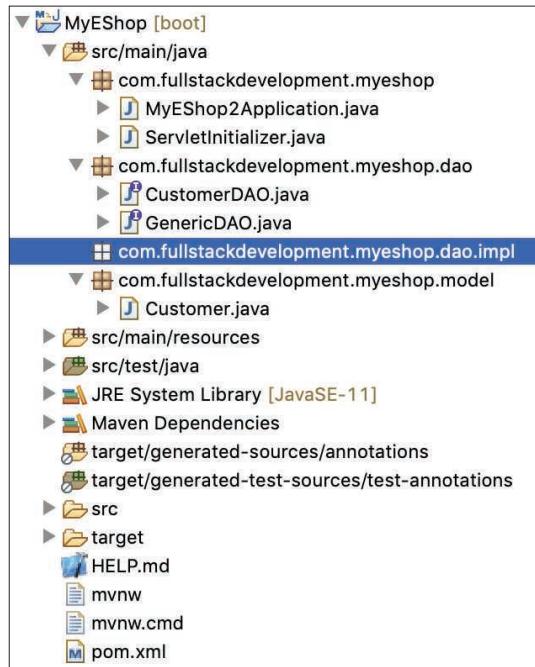


Figure 22.28 Newly package created.

We will add all the DAO implementation classes in this package. This is just to separate the implementation classes and interfaces for code readability purpose. Now, let us add our CustomerDAOImpl class by right clicking on this new package and selecting New->Class as shown in Figure 22.29.

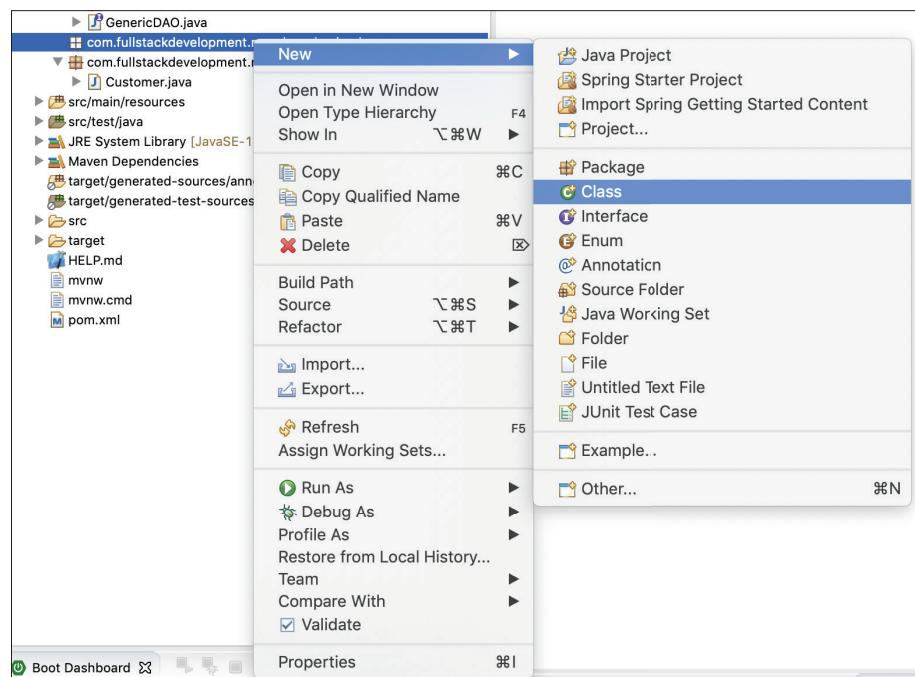


Figure 22.29 Menu to create a new class.

Once you click on the New->Class menu, you will be presented with a window to provide the information about the class as shown in Figure 22.30.

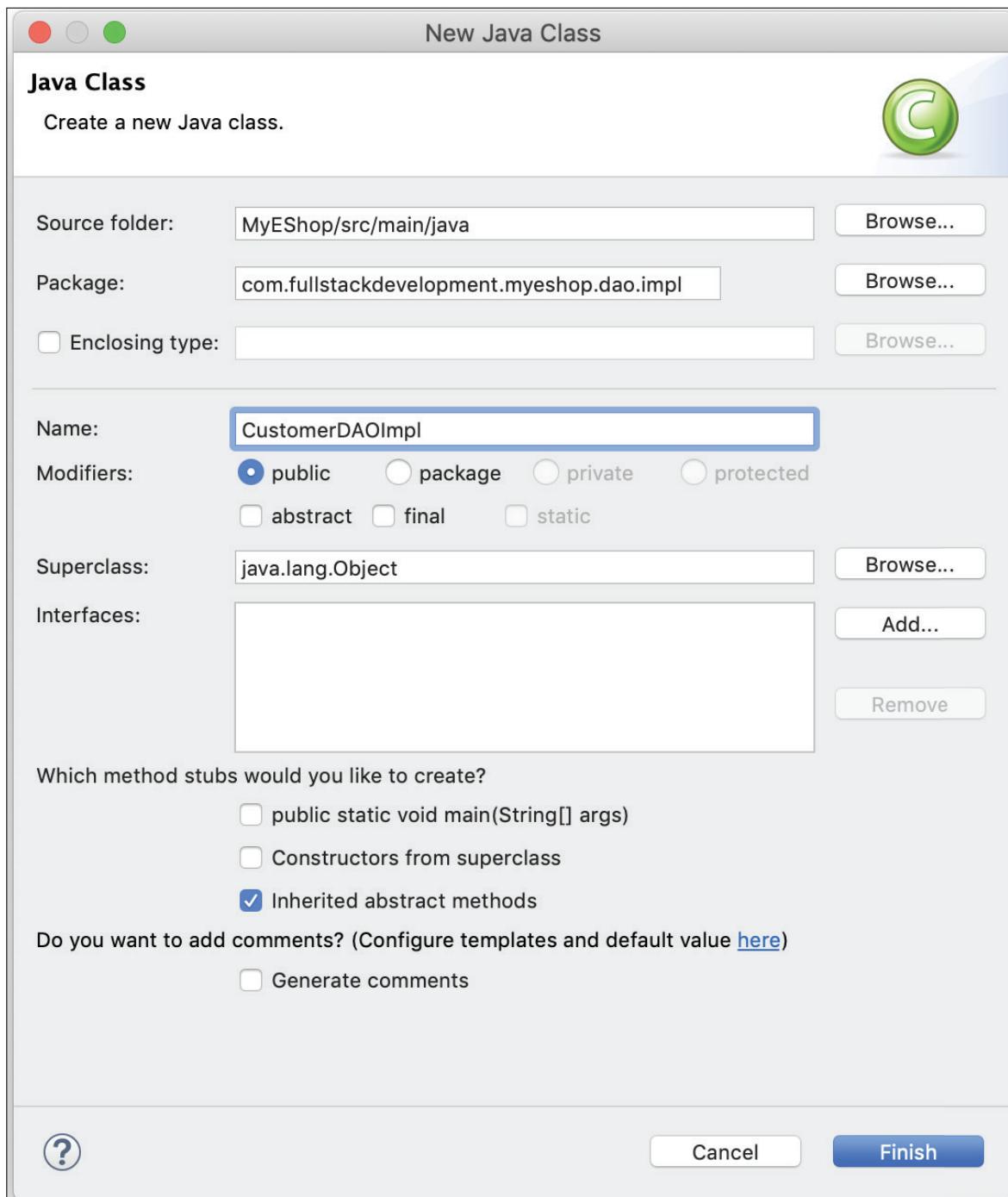


Figure 22.30 New class detail window.

In the name field type “CustomerDAOImpl” and click on the Finish button. This will add CustomerDAOImpl class under “com.fullstackdevelopment.myeshop.dao.impl” as shown in Figure 22.31.

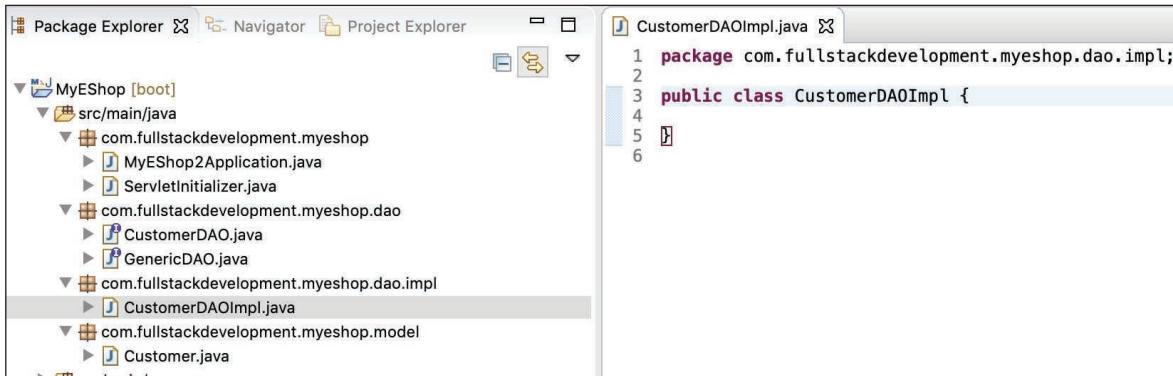


Figure 22.31 Newly created class.

Now, it is time to implement the CustomerDAO interface that we have created earlier. This interface will require us to provide a concrete implementation of the methods that are declared in the interface.

The screenshot shows the code editor for 'CustomerDAOImpl.java'. The code now includes the 'implements' keyword:

```

1 package com.fullstackdevelopment.myeshop.dao.impl;
2
3 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
4
5 public class CustomerDAOImpl implements CustomerDAO{
6
7 }
8

```

Figure 22.32 Code to show newly created class.

The red underline on the class name 'CustomerDAOImpl' in Figure 22.32 is because of the unimplemented methods that are declared in the CustomerDAO interface. IDE makes it easier for us by complaining about it by the moment we add implements keyword with the interface name (Figure 22.33). If you take the mouse pointer to the 'CustomerDAOImpl' word, you will see the details.

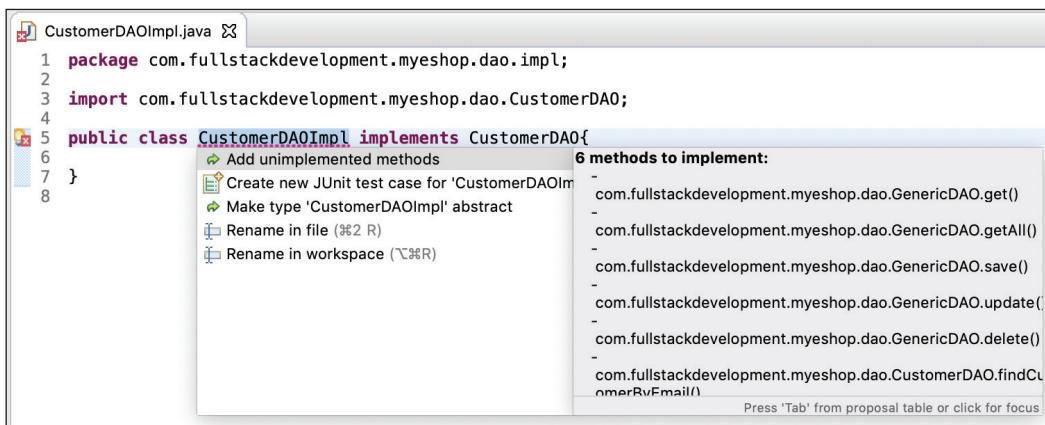


Figure 22.33 Auto-suggest dropdown to add unimplemented methods.

You can simply click on the “Add unimplemented methods” to add all the required methods’ skeleton code. Later we can add the actual code.

```

1 package com.fullstackdevelopment.myeshop.dao.impl;
2
3 import java.util.List;
4
5 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
6 import com.fullstackdevelopment.myeshop.model.Customer;
7
8 public class CustomerDAOImpl implements CustomerDAO{
9
10    @Override
11    public Customer get(int id) {
12        // TODO Auto-generated method stub
13        return null;
14    }
15
16    @Override
17    public List<Customer> getAll() {
18        // TODO Auto-generated method stub
19        return null;
20    }
21
22    @Override
23    public Long save(Customer t) {
24        // TODO Auto-generated method stub
25        return null;
26    }
27
28    @Override
29    public void update(Customer t) {
30        // TODO Auto-generated method stub
31    }
32
33
34    @Override
35    public void delete(Customer t) {
36        // TODO Auto-generated method stub
37    }
38
39
40    @Override
41    public Customer findCustomerByEmail(String email) {
42        // TODO Auto-generated method stub
43        return null;
44    }
45
46}
47

```

As you can see in the code in image above, it has added all the methods that are declared in the GenericDAO interface and one method “findCustomerByEmail” from CustomerDAO interface. This shows how easy it is to use an OOP language like Java.

**QUICK
CHALLENGE**

Create concrete DAO classes for the DAO interfaces you have created in the earlier Quick Challenge given in Section 22.4.

Now, we will create controller classes. We will come back and visit this DAO section in Chapter 23 when we will implement Hibernate code.

22.5 | Creating Controller



In this section, we will create controller classes for the models we have created earlier in Section 22.3. We will create a new package for these classes. So right click on the project and add a new package the way we have added earlier and give name it “com.fullstackdevelopment.myeshop.controller” as shown in Figure 22.34.

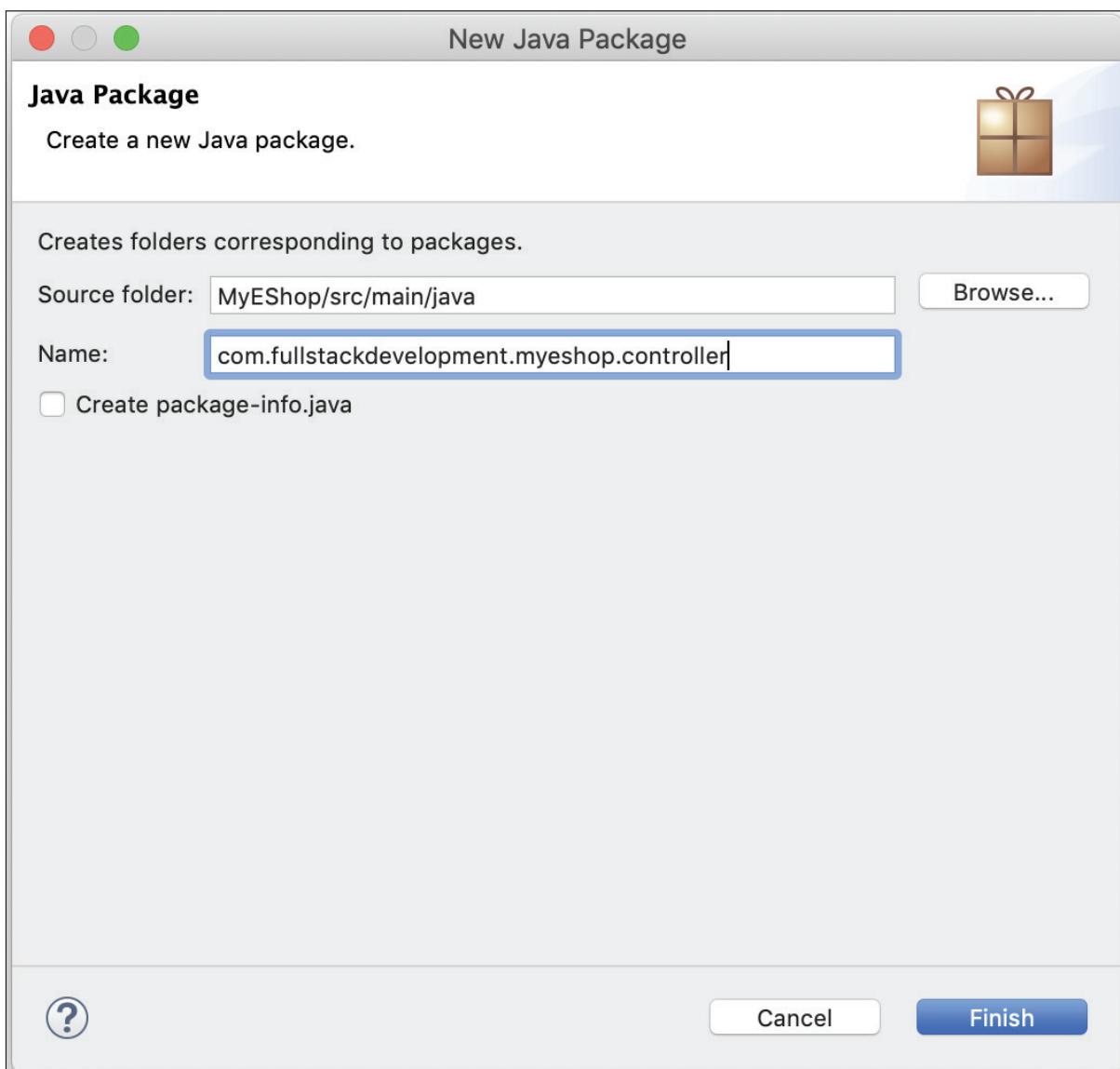


Figure 22.34 New package detail window.

Once you click on the Finish button, it will create “com.fullstackdevelopment.myeshop.controller”, package which will be listed in the project explorer as shown in Figure 22.35.

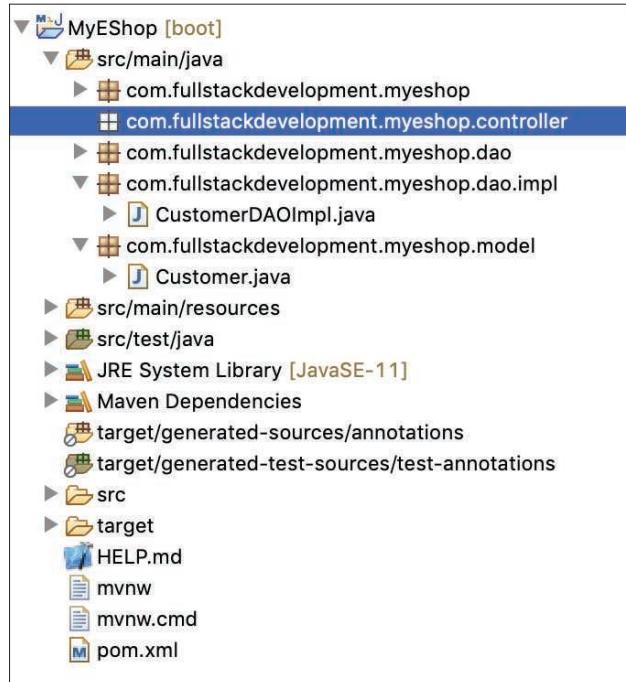


Figure 22.35 Newly created package.

We can now add the CustomerController class in this package. You can add it in the same way we have added the earlier classes by right clicking on the package and selecting New->Class as shown in Figure 22.36.

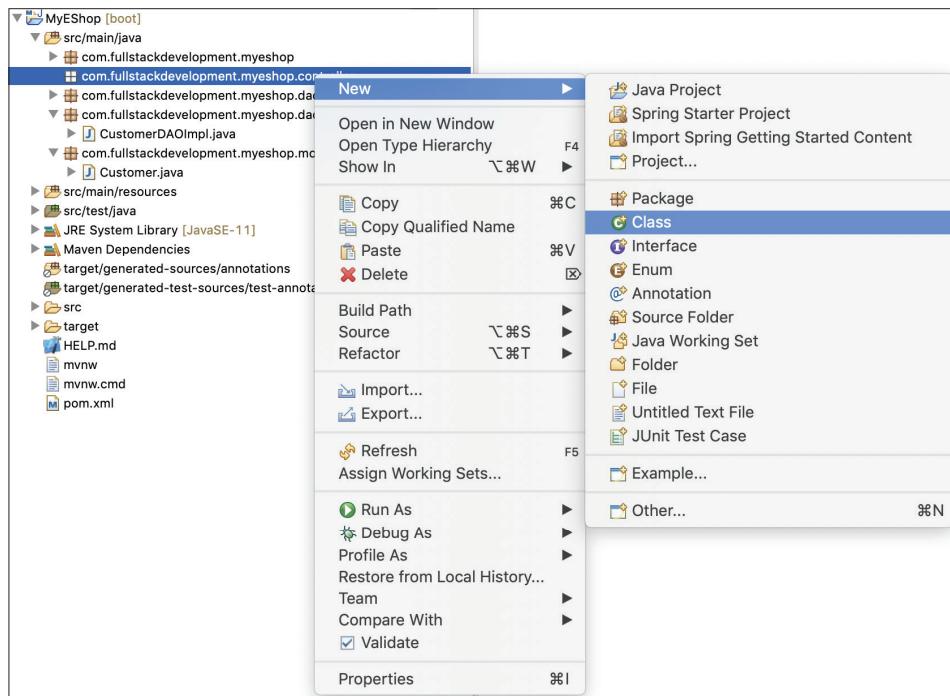


Figure 22.36 Menu to create a new class.

Once you click on this menu, it will present a window to enter the class details. Enter “CustomerController” in the name field as shown in Figure 22.37.

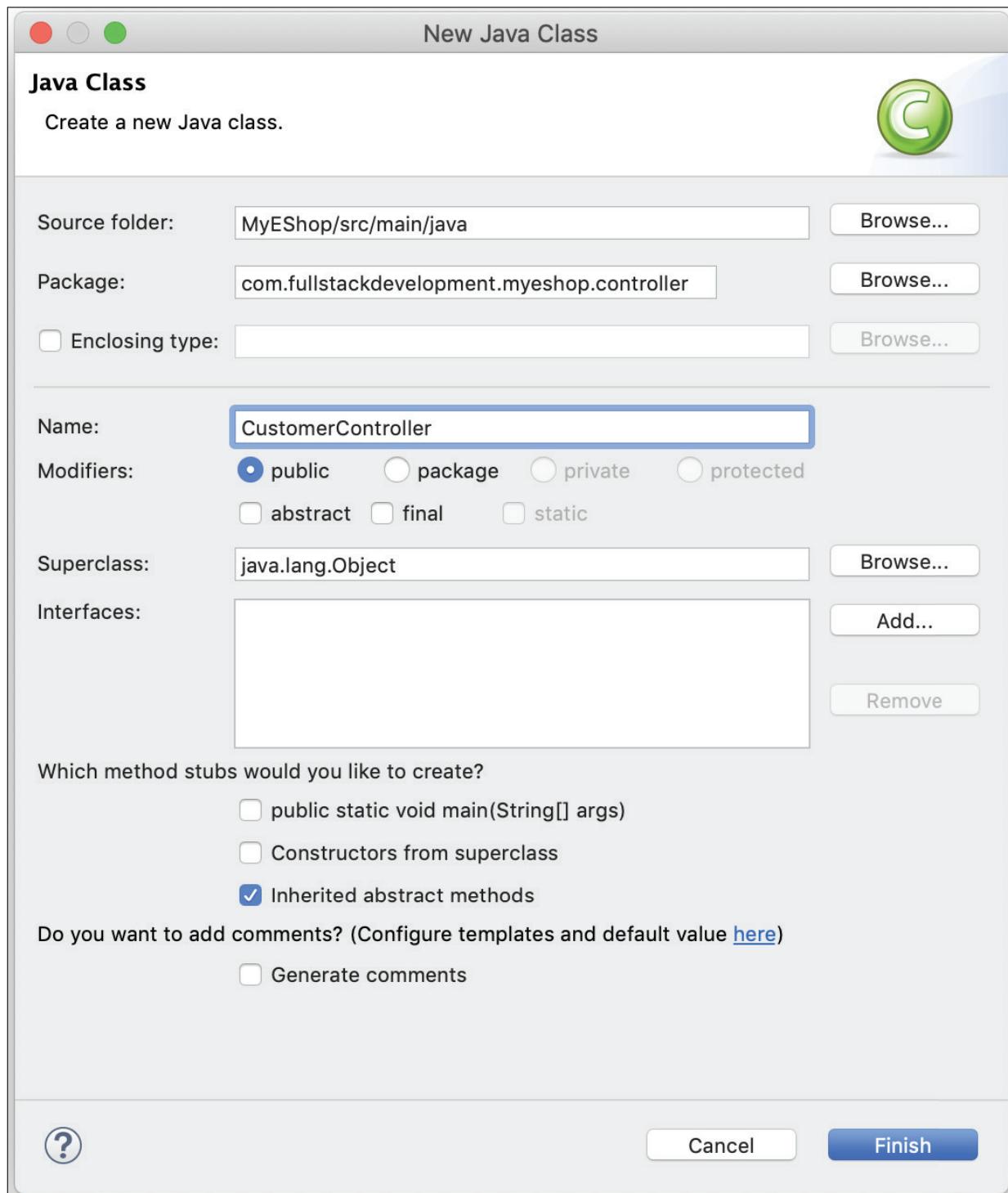


Figure 22.37 New Class Detail window.

Upon clicking on the Finish button, it will create a class named CustomerController under the package “com.fullstackdevelopment.myeshop.controller” as shown in Figure 22.38.

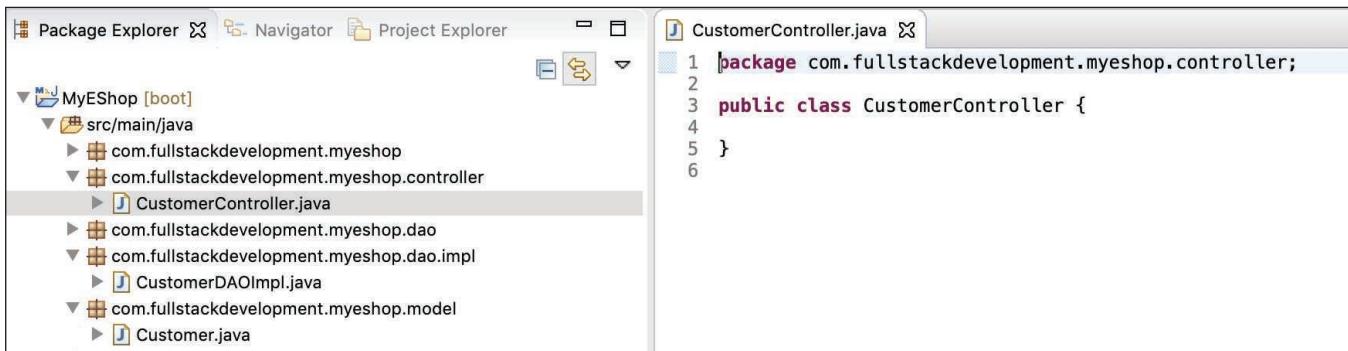


Figure 22.38 Newly created Class Code.

Now, we can add the controller code in this class. In order to tell Spring that the class we are creating is a controller, we need to add `@Controller` annotation from "org.springframework.stereotype.Controller". See the following code in image below.

The screenshot shows the code editor with the file 'CustomerController.java'. The code has been updated to include the `@Controller` annotation:

```

1 package com.fullstackdevelopment.myeshop.controller;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller
6 public class CustomerController {
7
8 }
9

```

Since we will be accessing Customer database through the DAO we have created earlier, we need to access CustomerDAO object in the controller. So, let us declare the CustomerDAO object.

The screenshot shows the code editor with the file 'CustomerController.java'. The code now includes the declaration of a private CustomerDAO object:

```

1 package com.fullstackdevelopment.myeshop.controller;
2
3 import org.springframework.stereotype.Controller;
4
5 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
6
7 @Controller
8 public class CustomerController {
9
10     private CustomerDAO customerDAO;
11
12 }
13

```

However, we have not initialized customerDAO object yet. We need to initialize it in order to use it in our code. Here, you can see how Spring uses Dependency Injection (DI) to get us the object we need without us initializing and managing its life cycle. We need to tell Spring to get us the CustomerDAO by using `@Autowired` annotation. This annotation tells Spring to manage the CustomerDAO object life cycle for us. See the following code in image below to know how to add `@Autowired` annotation.



```

1 package com.fullstackdevelopment.myeshop.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5
6 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
7
8 @Controller
9 public class CustomerController {
10
11     @Autowired
12     private CustomerDAO customerDAO;
13
14 }
15

```

@Autowired annotation is located in “org.springframework.beans.factory.annotation.Autowired”. This gives us the object we need to access the Customer database via CustomerDAO. Now, Spring will take care of the life cycle and we do not have to worry about managing customerDAO object at all. We can simply access it and it will be available for us without us doing anything.

Now, let us add a controller method which will be an endpoint API for our REST web service. We will start with a simple step to understand what we are doing in order to create this endpoint. The first step is to create a method body. In our application, we will need to get customer data by using his/her email address. So, let us add this method.



```

1 package com.fullstackdevelopment.myeshop.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5
6 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
7 import com.fullstackdevelopment.myeshop.model.Customer;
8
9 @Controller
10 public class CustomerController {
11
12     @Autowired
13     private CustomerDAO customerDAO;
14
15     public Customer findCustomerByEmail(String email) {
16         Customer customer = null;
17
18         return customer;
19     }
20 }
21

```

Please note that this is just a skeleton method we have added. We have not accessed the CustomerDAO yet to get the customer information. We have simply initialized an object and declared it null. Now, it is the time to access CustomerDAO to get the required data.

```

CustomerController.java ✎
1 package com.fullstackdevelopment.myeshop.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5
6 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
7 import com.fullstackdevelopment.myeshop.model.Customer;
8
9 @Controller
10 public class CustomerController {
11
12     @Autowired
13     private CustomerDAO customerDAO;
14
15     public Customer findCustomerByEmail(String email) {
16
17         Customer customer = customerDAO.findCustomerByEmail(email);
18         return customer;
19
20     }
21 }
22

```

Here, we have replaced the null declaration with customerDAO.findCustomerByEmail(email). Since Spring is managing customerDAO object for us, we will simply get the data we need by executing this line of code. Now, you can see how easy it is to use DI in your code.

Also note that in this chapter we are not doing anything related to Hibernate, so the data will not come from the database yet. In Chapter 23, we will have the fully working method when we wire the database code using hibernate.

We are not done yet as Spring does not know if we want findCustomerByEmail(String email) to be an endpoint. We have to explicitly tell Spring by adding @RequestMapping annotation. This annotation maps HTTP request with our handler method, which turns into an endpoint. In @RequestMapping we can define the parameters we are expecting in the method. In our case, we are looking for email parameter. So, let us see how to add this to the annotation.

```

CustomerController.java ✎
1 package com.fullstackdevelopment.myeshop.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestMethod;
7
8 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
9 import com.fullstackdevelopment.myeshop.model.Customer;
10
11 @Controller
12 public class CustomerController {
13
14     @Autowired
15     private CustomerDAO customerDAO;
16
17     @RequestMapping(value = "/customer/getCustomerByEmail/{email}", method = RequestMethod.GET)
18     public Customer findCustomerByEmail(String email) {
19
20         Customer customer = customerDAO.findCustomerByEmail(email);
21         return customer;
22
23     }
24 }
25

```

In the above image, you can see we have added `@RequestMapping(value = "/customer/getCustomerByEmail/{email}", method = RequestMethod.GET)` to define `findCustomerByEmail(String email)` as an endpoint which accepts email parameter as GET. In this example, “value = “/customer/getCustomerByEmail/{email}”,” defines the REST endpoint URL which will be used by our front end to access the Customer service. The next attribute `method = RequestMethod.GET` defines the request method we are using to get the parameter.

However, we are still not yet done as we need to map the incoming parameter to the parameter in the method. For this, we can use `@PathVariable("email")`. See the code in the following image.

```

CustomerController.java
1 package com.fullstackdevelopment.myeshop.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.web.bind.annotation.PathVariable;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RequestMethod;
8
9 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
10 import com.fullstackdevelopment.myeshop.model.Customer;
11
12 @Controller
13 public class CustomerController {
14
15     @Autowired
16     private CustomerDAO customerDAO;
17
18     @RequestMapping(value = "/customer/getCustomerByEmail/{email}", method = RequestMethod.GET)
19     public Customer findCustomerByEmail(@PathVariable("email") String email) {
20
21         Customer customer = customerDAO.findCustomerByEmail(email);
22         return customer;
23     }
24 }
25
26

```

Summary

In this chapter, we have learned creating Spring MVC based REST web service. We have taken a step-by-step approach from setting up the development environment, creating Spring Boot project to creating endpoints for the front-end applications to consume. We have seen the use of Dependency Injection and how easy it is to let Spring manage the object lifecycle for us. We have explored the Data Access Object design pattern to create an interface for accessing database without exposing the underline persistence layer. We have also seen the use of creating a generic interface, which can be extended by all interfaces to get some generic methods to avoid duplication of code. In addition, we learned about a few useful annotations like `@Autowired`, `@RequestMapping`, and `@PathVariable`.

In Chapter 23, we will see some practical use of Hibernate to wire database operations to our model and DAO. You will explore new concepts such as `@OneToOne` and `@OneToMany` relationships, and CRUD operations such as `save`, `saveOrUpdate`, etc.

Multiple-Choice Questions

1. Which of the following annotations is used to add dependency?
 (a) `@RequestMapping`
 (b) `@Entity`
 (c) `@Autowired`
 (d) `@PathVariable`
2. What is the full form of MVC?
 (a) Model View Controller
 (b) Model View Class
 (c) Meta Vector Class
 (d) Master View Class

3. Which of the following is a use of Package?
 - (a) To start a program
 - (b) Container for classes
 - (c) It looks good to start with package word
 - (d) To avoid a compiler error
4. Which of the following is a use of DAO?
 - (a) Provide access to an underlying database
 - (b) A class to contain business logic
 - (c) A class to contain API methods
 - (d) Just another class

Review Questions

1. How do you download and install Spring Tool Suite IDE?
2. How do you create a new project in Spring Tool Suite IDE?
3. How do you create a model in Spring Tool Suite IDE?
4. Which annotation do you need to add on a class to tell Spring that it is a model?
5. How do you create Data Access Object Interface and implementation class?
6. Which annotation do you use to add on a class to tell Spring that it is a controller?
7. What is the role of a Data Access Object class?
8. What is the role of a model?
9. What is the role of a controller?
10. What is the use of `@Autowired` annotation?
11. What is the use of `@RequestMapping` annotation and how do you use it on a method?
12. How do you accept parameters in a controller method?
13. How do you specify if the method accepts GET or POST?
14. How do you accept the parameters in a controller method?

Exercises

1. Create a simple service which can accept two integers and return their multiplication.
2. Define the use of `@Autowired`, `@RequestMapping`, and `@PathVariable`.

Project Idea

Take an example of a restaurant table book application. Identify the required entities and APIs that you will need to allow users to book a table. Based on this, create model and

controller classes which can expose these APIs and connect to the database.

Recommended Readings

1. Luciano Manelli. 2016. *Developing a Java Web Application in a Day: Step-by-step explanations with Eclipse, Tomcat and MySQL – A complete Java Project with Source Code (Java Web Programming Book 2)*. Luciano Manelli
2. Jeff Mcaffer. 2005. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications*. Addison-Wesley: Massachusetts
3. John R Hines. 2016. *Eclipse 4 Introduction Course for Java Developer: A brief introduction to Java programming using Eclipse (Eclipse Development Book 2)*.

Develop Models with Hibernate

LEARNING OBJECTIVES

After completing this chapter, you will be able to understand:

- How to wire hibernate to access database.
- How to perform CRUD operation on database with DAO.
- Basic yet important methods such as save and saveOrUpdate.

23.1 | Installing MySQL



We first need to create a database in which we can create the tables we need. For this example, we are using the MySQL database so you need to install it on your local computer. If you have not installed and setup MySQL, you can use the local server environment that will help you, and install and manage it with ease. Depending on your operating system, you could use one of the versions of MySQL explained in Sections 23.1.1 and 23.1.2. Please go through their documentation to learn about how to use them to set up MySQL.

23.1.1 Windows

WampServer: WampServer is the most useful local server environment for Windows users. It is free and opensource. It provides installation and management for Apache2, PHP, and MySQL.

Download: <http://www.wampserver.com/en/>
Documentation: <http://www.wampserver.com/en/>

HeidiSQL: This is a simple-to-use MySQL client for windows which allows you to manage your database easily. It can connect to local as well as remote servers. HeidiSQL offers a lot of features and can connect to many database systems such as MariaDB, MySQL, Microsoft SQL, and PostgreSQL.

Download: <https://www.heidisql.com/download.php>
Documentation: <https://www.heidisql.com/help.php>

23.1.2 Windows/Mac

MAMP: This is the Mac counterpart of the WampServer. Although different companies developed them, they have similar features in terms of managing local servers. MAMP is free to use. MAMP also has a Windows version so you could use this if you like this interface better than WampServer.

Download: <https://www.mamp.info/en/downloads/>
 Documentation: <https://documentation.mamp.info/>

Once you setup MySQL on your local machine, if you are not comfortable using command line, you may use one of the following free graphical user interface (GUI) applications to create database and tables.

Sequel pro: This is a free to use MySQL client. It offers a simple to use interface and good features. Although it does not offer the most advanced features, it is alright for small projects.

Download: <https://sequelpro.com/download>
 Documentation: <https://sequelpro.com/docs>

23.2 | Create Database and Tables



After setting up MySQL and your desired GUI, it is time to create our database. For our eShop example, we will give the same name to our database as our application. So create a database using the GUI and give it a name “MyEShop” as shown in Figure 23.1.

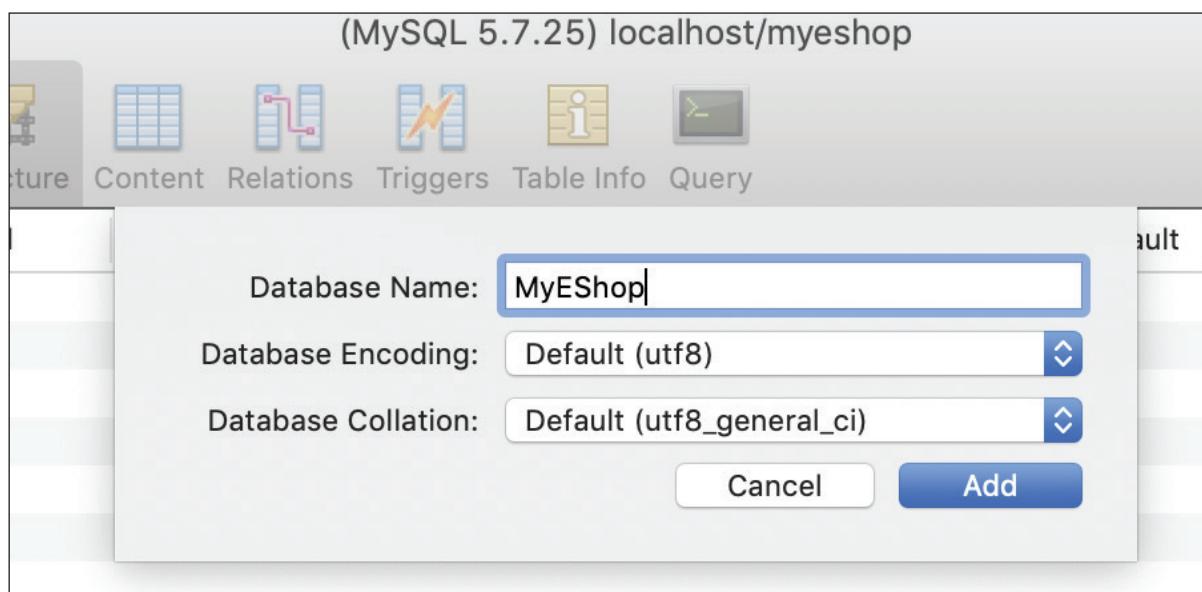
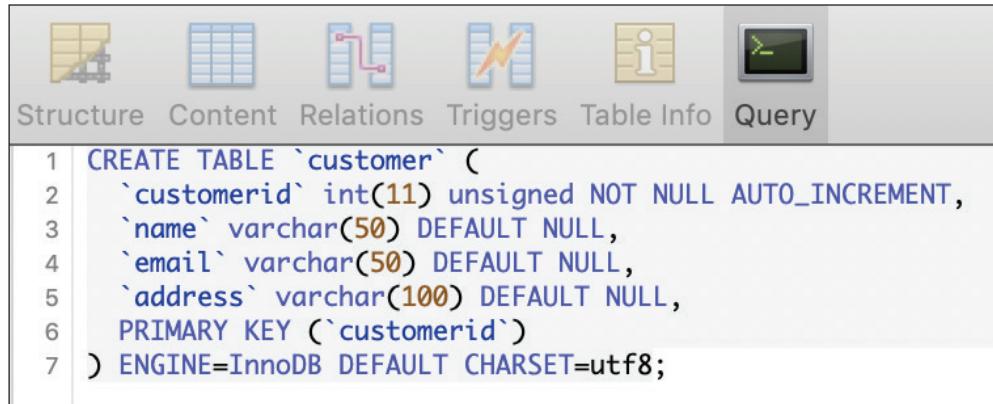


Figure 23.1 Adding a new database in GUI.

Once you click on the Add button, GUI will create the database for you. Now, it is time to add a table. You can either use the GUI to add it or use the following SQL which you can run in the Query window to create the Customer table with the fields we need.

```
CREATE TABLE `customer` (
  `customerid` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(50) DEFAULT NULL,
  `email` varchar(50) DEFAULT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`customerid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

See the following screenshot in Figure 23.2 to see how to run this query in the GUI.



```

CREATE TABLE `customer` (
  `customerid` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(50) DEFAULT NULL,
  `email` varchar(50) DEFAULT NULL,
  `address` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`customerid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

Figure 23.2 Query window to run CREATE TABLE command.

Please note that every table should have id column so we can identify the record and link it to other tables as foreign key. In this case, we will be more specific and call our id column as “customerid”. This is the field we have not added in our Customer model yet so we need to update our model to accommodate this field.

23.2.1 Linking Tables to Models

We will add customerid field as shown in the code in the following image, and let Spring know that this is an id by adding @Id annotation from javax.persistence package.



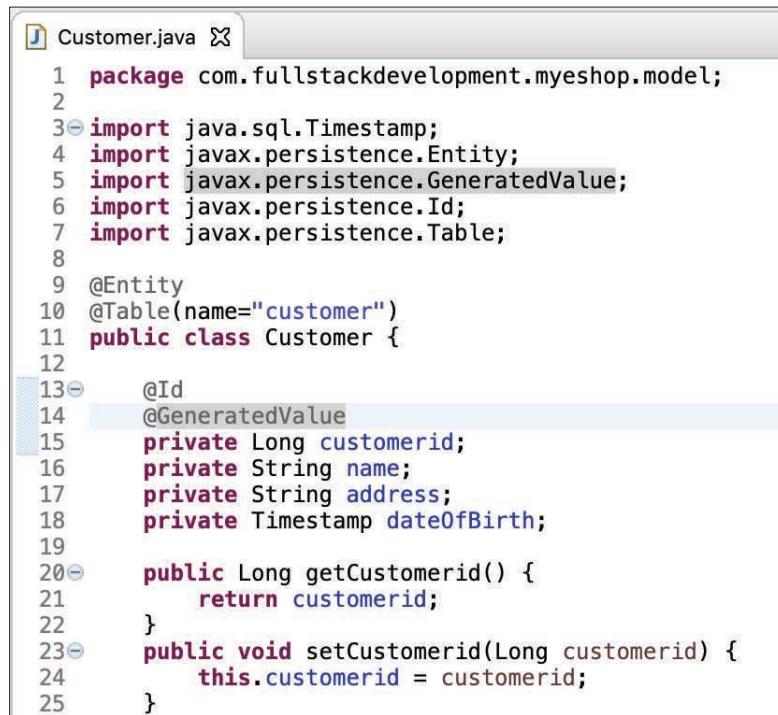
```

Customer.java

1 package com.fullstackdevelopment.myeshop.model;
2
3 import java.sql.Timestamp;
4 import javax.persistence.Entity;
5 import javax.persistence.Id;
6 import javax.persistence.Table;
7
8 @Entity
9 @Table(name="customer")
10 public class Customer {
11
12     @Id
13     private Long customerid;
14     private String name;
15     private String address;
16     private Timestamp dateOfBirth;
17
18     public Long getCustomerid() {
19         return customerid;
20     }
21     public void setCustomerid(Long customerid) {
22         this.customerid = customerid;
23     }

```

Since this is an id column, we would like it to be incremental. This is what we need to tell Hibernate as well. For this, we will be using @GeneratedValue annotation from javax.persistence package.



```

1 package com.fullstackdevelopment.myeshop.model;
2
3 import java.sql.Timestamp;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.Id;
7 import javax.persistence.Table;
8
9 @Entity
10 @Table(name="customer")
11 public class Customer {
12
13     @Id
14     @GeneratedValue
15     private Long customerid;
16     private String name;
17     private String address;
18     private Timestamp dateOfBirth;
19
20     public Long getCustomerid() {
21         return customerid;
22     }
23     public void setCustomerid(Long customerid) {
24         this.customerid = customerid;
25     }

```



How do the values in customerid table get stored if @GeneratedValue annotation is removed?

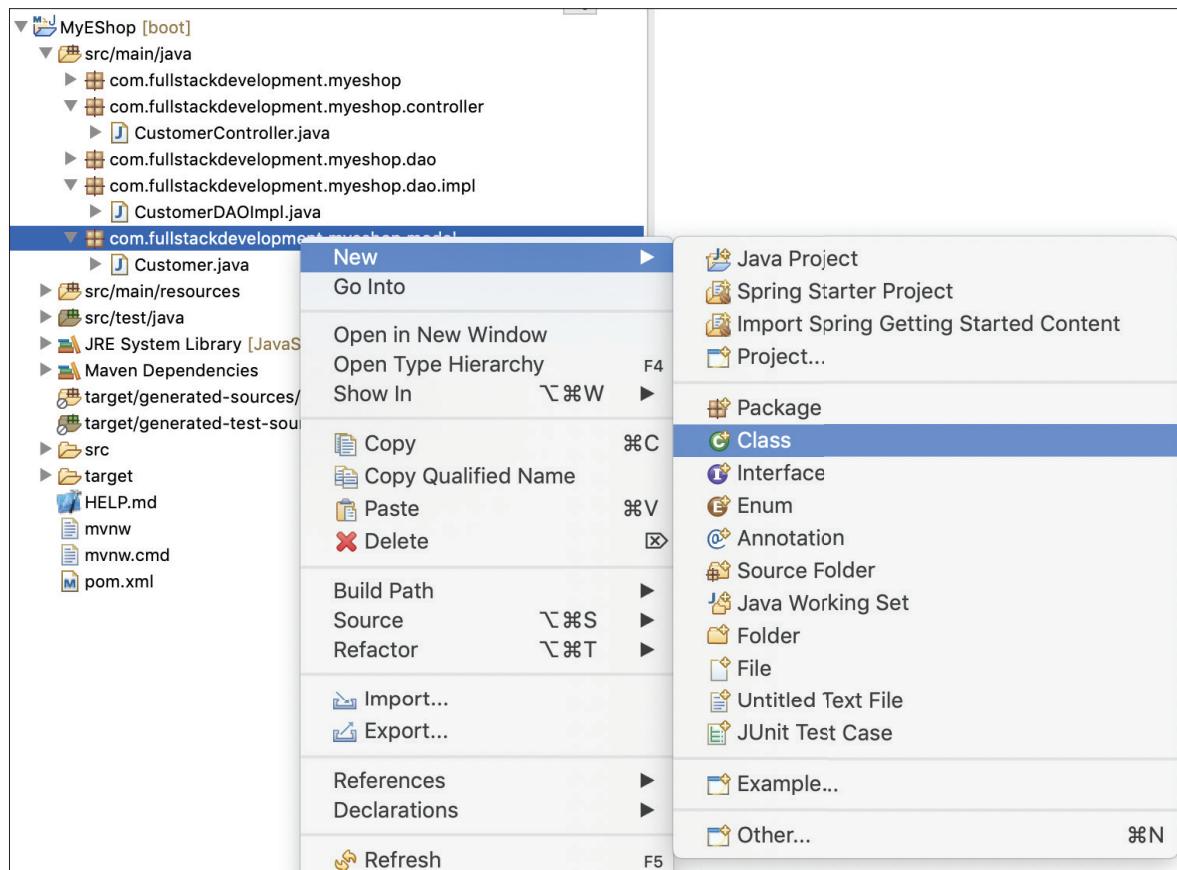


Figure 23.3 Menu to create a new class.

There is one more important step to map this model to the customer table we have created. This step we have already done in Chapter 22 by adding `@Table` annotation to our `Customer` class name. We also specified the table name that is mapped to this model by using name attribute `@Table(name="customer")`.

In Chapter 22 we talked about making address as a separate entity so we can store multiple addresses for the customer. So, let us work on that now. We need to update our `Customer` table and model for this. Let us first create a new model called `Address`, with basic address related fields. Right-click on the “`com.fullstackdevelopment.myeshop.model`” package and click on `New->Class`.

After clicking on this menu, you will see a window to add details about the model. Let us call this class as “`Address`” as shown in Figure 23.4.

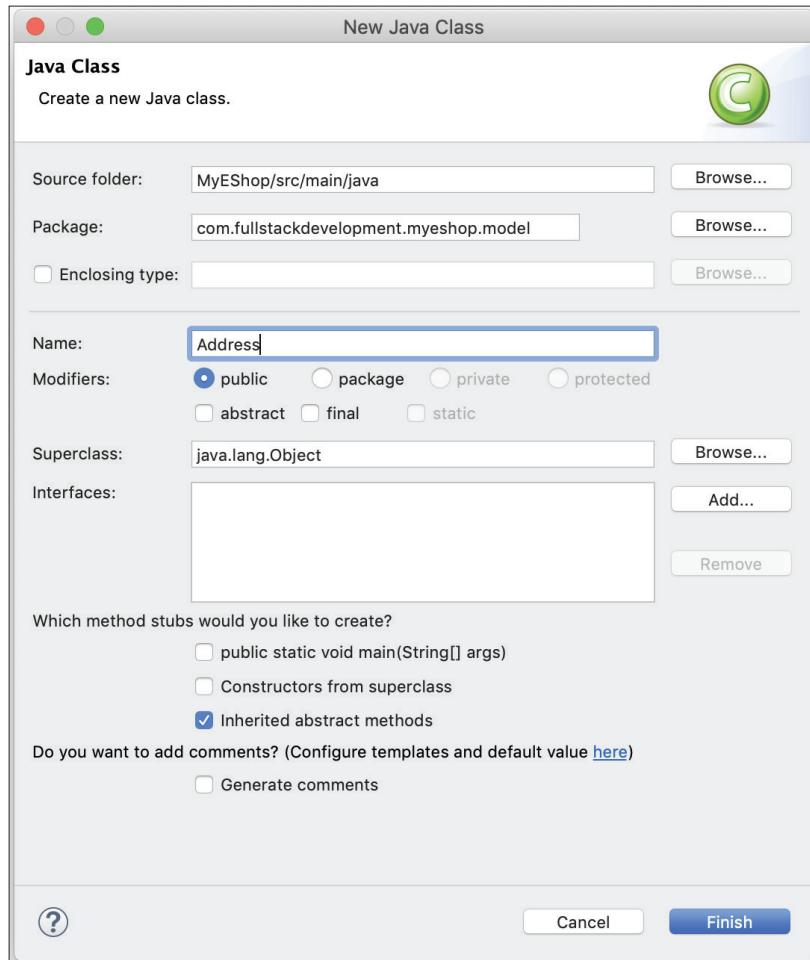


Figure 23.4 New class detail window.

Upon clicking on the `Finish` button, it will create the `Address` class under “`com.fullstackdevelopment.myeshop.model`” package as shown in Figure 23.5.

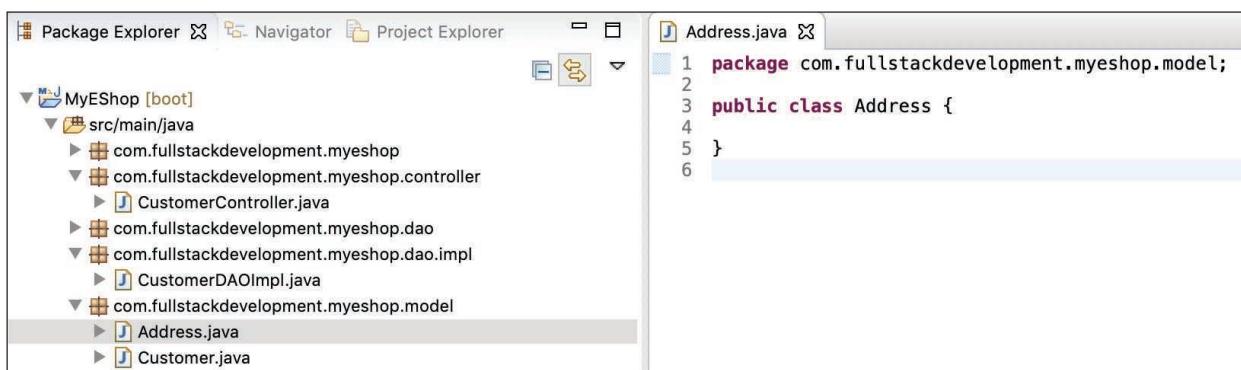
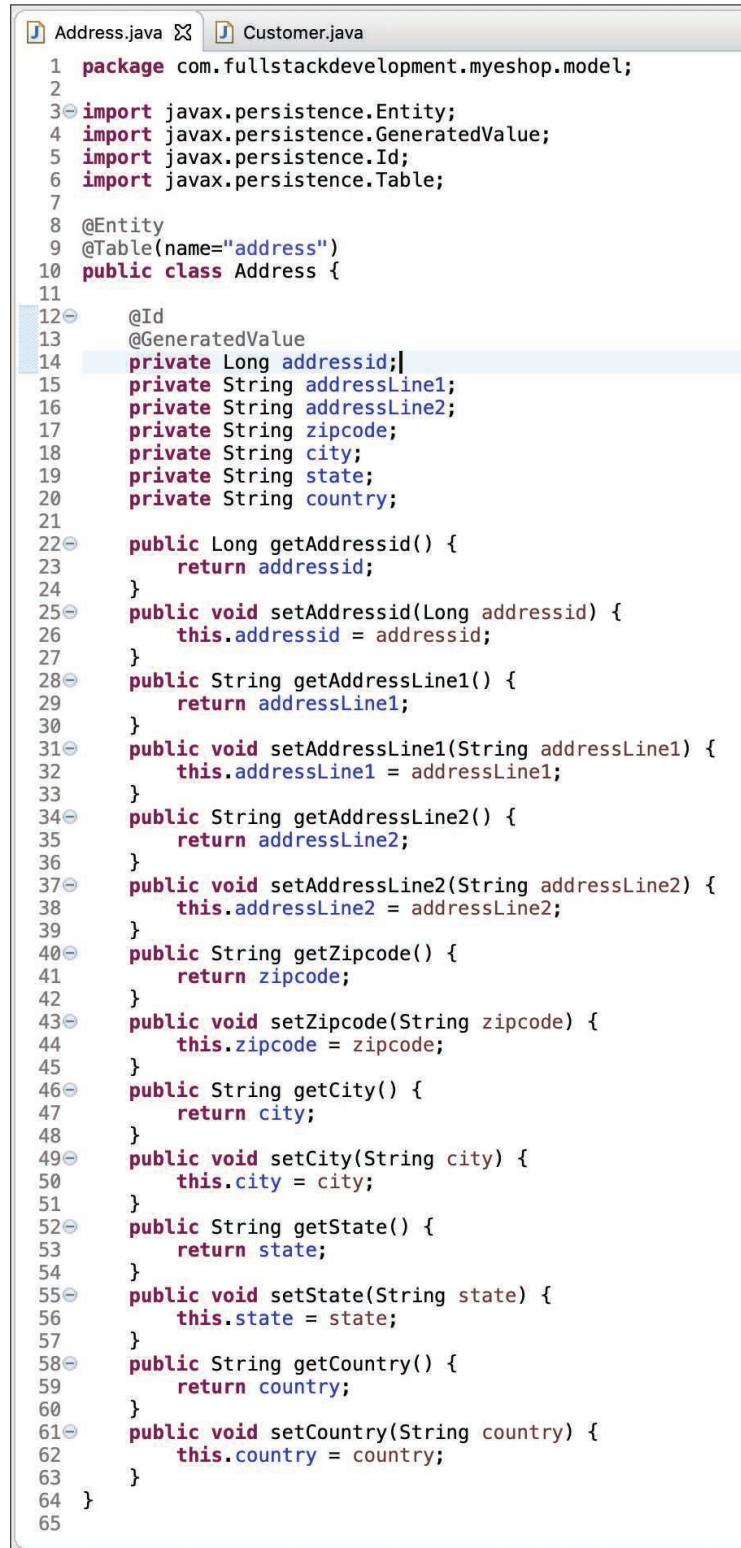


Figure 23.5 Newly added `Address` class.

Once the class is created, add basic address related fields and “addressid” as an id field with the required annotations.



```

1 package com.fullstackdevelopment.myeshop.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.Id;
6 import javax.persistence.Table;
7
8 @Entity
9 @Table(name="address")
10 public class Address {
11
12     @Id
13     @GeneratedValue
14     private Long addressid;
15     private String addressLine1;
16     private String addressLine2;
17     private String zipcode;
18     private String city;
19     private String state;
20     private String country;
21
22     public Long getAddressid() {
23         return addressid;
24     }
25     public void setAddressid(Long addressid) {
26         this.addressid = addressid;
27     }
28     public String getAddressLine1() {
29         return addressLine1;
30     }
31     public void setAddressLine1(String addressLine1) {
32         this.addressLine1 = addressLine1;
33     }
34     public String getAddressLine2() {
35         return addressLine2;
36     }
37     public void setAddressLine2(String addressLine2) {
38         this.addressLine2 = addressLine2;
39     }
40     public String getZipcode() {
41         return zipcode;
42     }
43     public void setZipcode(String zipcode) {
44         this.zipcode = zipcode;
45     }
46     public String getCity() {
47         return city;
48     }
49     public void setCity(String city) {
50         this.city = city;
51     }
52     public String getState() {
53         return state;
54     }
55     public void setState(String state) {
56         this.state = state;
57     }
58     public String getCountry() {
59         return country;
60     }
61     public void setCountry(String country) {
62         this.country = country;
63     }
64 }
65

```

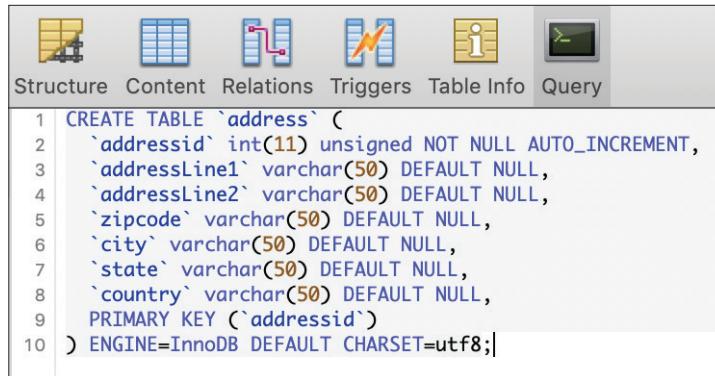
As you can see, we have added the basic fields and `@Entity` annotation along with `@Table` annotation and specified our table as “address”. Now, let us create this table in the database as well. You can run the following SQL in the Query window of the GUI.

```

CREATE TABLE `address` (
    `addressid` int(11) unsigned NOT NULL AUTO_INCREMENT,
    `addressLine1` varchar(50) DEFAULT NULL,
    `addressLine2` varchar(50) DEFAULT NULL,
    `zipcode` varchar(50) DEFAULT NULL,
    `city` varchar(50) DEFAULT NULL,
    `state` varchar(50) DEFAULT NULL,
    `country` varchar(50) DEFAULT NULL,
    PRIMARY KEY (`addressid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

See the following image to see how to run this query.



QUICK CHALLENGE

Based on the entity relationship diagram, create SQL for other tables.

23.2.2 Setting up Relationships

Now, we need to link this table with Customer so Customer can store multiple addresses. For this, we need to update our SQL as well as our Model code. Let us first update our table to link address to customer. We can use “customerid” as a foreign key in the address table.

First add “customerid” as a field in the “address” table. You can use the following command for that.

```
ALTER TABLE `address` ADD COLUMN customerid int(11) not null;
```

Please note that we are not yet adding a database constraint as foreign key on the address table. We are just creating customerid field in the address table so hibernate can link these two tables for us.

Now, we need to update our Address class to add this newly added field. However, we will not add this field as Long but we will reference it to the Customer object. Later, we will see how to link these tables in the model code.

```

8  @Entity
9  @Table(name="address")
10 public class Address {
11
12     @Id
13     @GeneratedValue
14     private Long addressid;
15     private String addressLine1;
16     private String addressLine2;
17     private String zipcode;
18     private String city;
19     private String state;
20     private String country;
21     private Customer customer;
22

```

You will also need to add Getter and Setter for this field.

```

65④  public Customer getCustomer() {
66      return customer;
67  }
68④  public void setCustomer(Customer customer) {
69      this.customer = customer;
70  }

```

Now, it is time to let Spring know how we need to access Customer and Address. As discussed earlier, every customer can have multiple addresses such as Shipping, Billing, etc. Hence, the customer can have one-to-many relationship with address. Similarly, address can have many-to-one relationship with customer. This is what we need to define in the code so Spring will know how to treat these two entities.

Before we specify in both the models, we first need to add address field in the Customer model. Since we would like to define one-to-many relationship, we will add address as List so we can get multiple addresses for one customer entry. See the code in the following image.

```

11 @Entity
12 @Table(name="customer")
13 public class Customer {
14
15④  @Id
16  @GeneratedValue
17  private Long customerid;
18  private String name;
19  private String address;
20  private Timestamp dateOfBirth;
21  private List<Address> addresses;
22
23④  public List<Address> getAddresses() {
24      return addresses;
25  }
26④  public void setAddresses(List<Address> addresses) {
27      this.addresses = addresses;
28  }

```

Once we define this, we now need to define these relationships. For this, we can use @OneToMany and @ManyToOne annotations. See the Customer model code in the following image.

```

1 package com.fullstackdevelopment.myeshop.model;
2
3④ import java.sql.Timestamp;
4 import java.util.List;
5
6 import javax.persistence.CascadeType;
7 import javax.persistence.Entity;
8 import javax.persistence.FetchType;
9 import javax.persistence.GeneratedValue;
10 import javax.persistence.Id;
11 import javax.persistence.OneToMany;
12 import javax.persistence.Table;
13
14 @Entity
15 @Table(name="customer")
16 public class Customer {
17
18④  @Id
19  @GeneratedValue
20  private Long customerid;
21  private String name;
22  private String address;
23  private Timestamp dateOfBirth;
24
25④  @OneToMany(mappedBy="customer",fetch = FetchType.LAZY, cascade = CascadeType.MERGE)
26  private List<Address> addresses;
27

```

@OneToMany annotation has mappedBy property where we can define the variable reference from the Address class. Earlier, we have added field **private Customer customer;** in the address class. This is what we need to reference here.



What will happen if we remove @OneToMany annotation from the addresses field?

Then we have another property called fetch. This is an important property to set as it defines FetchType. In our case, we have used LAZY as the fetch type. It means Hibernate will not query tables until we ask for it specifically. This will speed up the fetch operation as Hibernate does not need to define join while running select query on the Customer table. If we explicitly ask to give customer addresses then Hibernate will add the join query. This helps in increasing application performance. The other fetch type is called EAGER. This fetch type instructs Hibernate to get addresses each time we ask for a customer object. It adds a little penalty on the performance side.

The last property that you see is “cascade”. This property defines what happens when a CRUD operation is performed on customer object. There are many cascade types you can use, as follows.

1. **CascadeType.PERSIST:** When used, `save()` or `persist()` operations cascade to related entities.
2. **CascadeType.MERGE:** When used on the owning entity, all the related entities of the owning entity are merged upon the owning entity is merged.
3. **CascadeType.REFRESH:** When used, related entities are refreshed.
4. **CascadeType.REMOVE:** When used, upon deletion of the owning entity, all the related entities get removed.
5. **CascadeType.DETACH:** When used, in case of “manual detach” all the related entities are detached.
6. **CascadeType.ALL:** When used, it performs all the above cascade operations.

In order to get it working properly, we also need to tell Hibernate about Address to Customer relationship. Hence, we need to define @ManyToOne relationship on the customer object in the Address class.

```

Address.java ✘ Customer.java
1 package com.fullstackdevelopment.myeshop.model;
2
3 import javax.persistence.CascadeType;
4 import javax.persistence.Entity;
5 import javax.persistence.FetchType;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.Id;
8 import javax.persistence.JoinColumn;
9 import javax.persistence.ManyToOne;
10 import javax.persistence.Table;
11 import com.fasterxml.jackson.annotation.JsonBackReference;
12
13 @Entity
14 @Table(name="address")
15 public class Address {
16
17     @Id
18     @GeneratedValue
19     private Long addressid;
20     private String addressLine1;
21     private String addressLine2;
22     private String zipcode;
23     private String city;
24     private String state;
25     private String country;
26
27     @JsonBackReference
28     @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.MERGE)
29     @JoinColumn(name="customerid", referencedColumnName = "customerid")
30     private Customer customer;
31

```

As you can see, we have added the following three lines on the Customer object declaration.

```
@JsonBackReference
@ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.MERGE)
@JoinColumn(name="customerid", referencedColumnName = "customerid")
```

This will tell Hibernate how to link address with customer.

We have already learned fetch and cascade, so let us jump on to the other two annotations. `@JoinColumn` annotation tells Hibernate about the column in the database that we need use for the link. This is the foreign key field we have created in the address table. The name property allows us to specify this field name from the table, in our case it is “customerid”. The `referencedColumnName` property allows us to specify the column name (primary key) of the customer class, which is linked with address via “customerid” field. There is one more annotation `@JsonBackReference` you can see on top of `@ManyToOne` annotation. This annotation is designed to solve the infinite recursion problem by linking two-way – one for Parent and other for Child.

Now, let us look at another type of relationship. When two entities are related to each other in a singular fashion, it is known as one-to-one relationship. This type of relationship is defined by `@OneToOne` annotation. The owning class field will have this annotation.

QUICK CHALLENGE

State the differences of `@OneToOne`, `@OneToMany`, and `@ManyToOne`.

23.3 | Making DAO to Perform CRUD



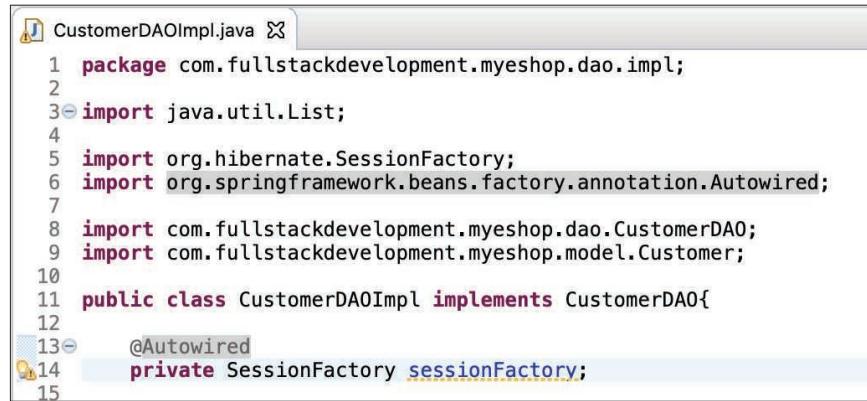
We can now move on to the DAO class section to write code that can perform CRUD operations on these models.

In our `CustomerDAOImpl` class, we will be adding this interface code, which will hide the model details and only expose it via `CustomerDAO` interface methods and hence further hides the implantation from the calling classes. Since we will be performing CRUD operations, we need to make sure the transactions are handled properly. For this, we can simply annotate the class with `@Transactional` annotation. This tells Spring that we need Spring's help to manage the transactions for this class.

```
CustomerDAOImpl.java
1 package com.fullstackdevelopment.myeshop.dao.impl;
2
3 import java.util.Collection;
4 import java.util.Optional;
5
6 import javax.transaction.Transactional;
7
8 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
9 import com.fullstackdevelopment.myeshop.model.Customer;
10
11 @Transactional
12 public class CustomerDAOImpl implements CustomerDAO{
13 }
```

If we asked Spring to manage transactions on our behalf by annotating the class as `@Transactional`, Spring will create a proxy class which is invisible at runtime and provides a way to inject behavior into the object before, after, or around method calls.

Now, we can move on to write code for `findCustomerByEmail(String email)` method. In order to use Hibernate to perform CRUD operations, we need `SessionFactory` from the Hibernate package. As shown in the code in the following image, we will use `@Autowired` annotation to inject `SessionFactory` to `CustomerDAOImpl` class.



```

1 package com.fullstackdevelopment.myeshop.dao.impl;
2
3 import java.util.List;
4
5 import org.hibernate.SessionFactory;
6 import org.springframework.beans.factory.annotation.Autowired;
7
8 import com.fullstackdevelopment.myeshop.dao.CustomerDAO;
9 import com.fullstackdevelopment.myeshop.model.Customer;
10
11 public class CustomerDAOImpl implements CustomerDAO{
12
13     @Autowired
14     private SessionFactory sessionFactory;
15

```

We also need to create a constructor and pass SessionFactory parameter, which will be configured in the XML configuration.

```

public CustomerDAOImpl(SessionFactory sessionFactory) {
    this.sessionFactory = sessionFactory;
}

```

Once we get the sessionFactory object, we can add the following code to fetch a Customer record by email. Before we proceed with this task, we need to add the email field in the Customer model class. We have already added this into the database, so it is time to add in the model.



```

1 package com.fullstackdevelopment.myeshop.model;
2
3 import java.sql.Timestamp;
4
5
6 @Entity
7 @Table(name="customer")
8 public class Customer {
9
10     @Id
11     @GeneratedValue
12     private Long customerid;
13     private String name;
14     private String email;
15     private Timestamp dateOfBirth;
16
17
18
19
20
21
22
23

```

As you can see from the code in the above image, we have removed the String type “address” field and added String type “email” field. We have also created getter and setter for this field. Let us proceed with adding customer search code with email address.



```

51     @Override
52     public Customer findCustomerByEmail(String email) {
53         CriteriaBuilder criteriaBuilder = sessionFactory.getCurrentSession().getCriteriaBuilder();
54         CriteriaQuery<Customer> criteriaQuery = criteriaBuilder.createQuery(Customer.class);
55         Root<Customer> root = criteriaQuery.from(Customer.class);
56         criteriaQuery.select(root).where(criteriaBuilder.equal(root.get("email"), email));
57         Query<Customer> q = sessionFactory.getCurrentSession().createQuery(criteriaQuery);
58         return q.getSingleResult();
59     }
60

```

In the above example, we have used CriteriaBuilder from “javax.persistence.criteria.CriteriaBuilder” package. It is accessible via sessionFactory. This CriteriaBuilder object is then used to get CriteriaQuery instance. Then the “from” method is used to set the Customer class as “root” on this query object. Now the root is set, we used “where” method to set the comparison on email column. Then we used create query object for this CriteriaQuery that we built. And finally, we run “q.getSingleResult()” to get the desired record for the email parameter we are passing to this method.

**QUICK
CHALLENGE**

With help of findCustomerByEmail(String email) example, write code for get(int id) and getAll methods.

Let us move on to write code for the save method. In Hibernate, there are three options we can use to save the record in the database.

1. **save():** As the name suggests, this is used to save an entity into the database. It also returns a generated identifier. If the entity already exists in the database, it throws an exception.
2. **persist():** It is very similar to save method. The only difference is it does not return a generated identifier but returns a void.
3. **saveOrUpdate():** This is useful when you do not know if the entity exists in the database. This method will either save or update the entity into the database. Hence, it does not throw an exception as `save()` method throws if the entity already exists in the database; it will simply update it.

The code in the following image shows how to save record through Hibernate and how easy it is to set data without writing a single line of SQL.

```

@Override
public Long save(Customer t) {
    Customer customer = t;

    if (customer == null) {
        customer = new Customer();
        customer.setName("Albert Einstein");
        customer.setEmail("Albert@Einstein.com");

        // set date of birth
        SimpleDateFormat format = new SimpleDateFormat("yyyyMMdd");
        try {
            Date parsed = format.parse("1879014");
            Timestamp dateOfBirth = new Timestamp(parsed.getTime());
            customer.setDateOfBirth(dateOfBirth);
        } catch (ParseException e) {
            System.out.println("Error in parsing the date of birth");
            e.printStackTrace();
        }
    }

    // set billing and shipping addresses
    List<Address> allAddresses = new ArrayList<Address>();

    Address shippingAddress = new Address();
    shippingAddress.setAddressLine1("112");
    shippingAddress.setAddressLine2("Mercer St");
    shippingAddress.setCity("Princeton");
    shippingAddress.setZipcode("NJ 08540");
    shippingAddress.setCountry("USA");
    sessionFactory.getCurrentSession().save(shippingAddress);
    allAddresses.add(shippingAddress);

    Address billingAddress = new Address();
    billingAddress.setAddressLine1("112");
    billingAddress.setAddressLine2("Mercer St");
    billingAddress.setCity("Princeton");
    billingAddress.setZipcode("NJ 08540");
    billingAddress.setCountry("USA");
    sessionFactory.getCurrentSession().save(billingAddress);
    allAddresses.add(billingAddress);

    customer.setAddresses(allAddresses);
}

sessionFactory.getCurrentSession().save(customer);
sessionFactory.getCurrentSession().flush();

return customer.getCustomerId();
}

```

In the above code, multiple steps are taking place. We are first checking if the parameter is null or not. If it is, say, null, we are initializing a Customer object. Now, we will use this customer object to set field values. If it is not null then we are directly saving it using the sessionFactory. In the null case, setting up name and email are straightforward processes. For setting up date of birth, we need to convert the given date into a Timestamp with the help of SimpleDateFormat. After that, we are declaring an ArrayList for Address entity. This ArrayList will hold all the addresses. We are creating a shippingAddress by initializing an Address object and setting up the field data. After that we are saving the address entity into the database with the help of sessionFactory. We are repeating the process for the billingAddress. Once the addresses are saved, we are adding this list to the customer object's addresses field and later saving it into the database. In the end, we are flushing the database so database is updated. Since `save()` method returns generatedId (in our case customerid), we are simply returning it.

QUICK CHALLENGE

By using save method example shown in Section 23.3, write code for update method.

Now, let us write code for “delete” method. There are two types of entities we will be dealing with when deleting from the database. One is transient and another one is persistence. Since the transient entity is not associated with the session, we use identifier to remove the instance from the database. For example, if Customer object is transient then we can delete it by specifying ID of it. See the following code block.

```
Customer customer = new Customer();
customer.setCustomerid(1L);
sessionFactory.getCurrentSession().delete(customer);
```

The above code will delete a Customer record which contains customerid = 1.

In case of persistence, we load the entity first and then delete it. We use sessionFactory object to load the entity.

```
Customer customer = sessionFactory.getCurrentSession().load(Customer.class, 1L);
if(customer != null) {
    sessionFactory.getCurrentSession().delete(customer);
}
```

In the above example, we are first loading Customer using sessionFactory and then making sure it is not null before executing delete operation on it.

In our case, since we are getting a customer object via parameter, we do not need to load this entity. We will just make sure that the entity is not null before running the delete operation on it.

```
@Override
public void delete(Customer t) {
    Customer customer = t;
    if(customer != null) {
        sessionFactory.getCurrentSession().delete(customer);
    }
}
```

Summary

In this chapter, we have learned how to install and use MySQL, create database and tables, and run queries using GUI. We have also seen how to use Hibernate to perform CRUD operations on a database. We have learned various relationships like @OneToOne and @OneToMany. We have also learned about performing CRUD operations on a database including the difference between save, persist, and saveOrUpdate.

Multiple-Choice Questions

1. What is RDBMS?
 - (a) Relational Database Management System
 - (b) Rational Data Batch Management System
 - (c) Relational Database Messaging System
 - (d) Rotational Database Management System
 2. Which of the following is not a valid cascade type?
 - (a) CascadeType.PERSIST
 - (b) CascadeType.MERGE
 - (c) CascadeType.SAVE
 - (d) CascadeType.REFRESH
 3. `save()` is used to update a record in Hibernate.
- (a) True
 - (b) False
4. SessionFactory is a thread safe object
 - (a) True
 - (b) False
 5. _____ annotation is used to make value of id column incremental.
 - (a) `@GeneratedValue`
 - (b) `@Id(Increment)`
 - (c) `@Id(ColumnValue = "Increment")`
 - (d) `@Incremental`

Review Questions

1. What is MySQL?
2. What is the use of Database?
3. What is the difference between DBMS and RDBMS?
4. How to make sure that the class is treated as model to map with a table in the database?
5. What is Hibernate?
6. What is object-relational mapping?
7. What are the disadvantages of using object-relational mapping?
8. What is the difference between `save()` and `saveOrUpdate()`?
9. When can you use `persist()`?
10. What is CriteriaBuilder?

Exercises

1. Define all the different Database encodings and their effect on table.
2. Define all the different Database collations and their effect on table.
3. Change the column name “name” to “fullname” of the table customer we created in section 23.2.
4. Explain the use of `@JsonBackReference` and the effect on the table if removed from the field.

Project Idea

Create a simple application to create a TODO list application. Use Hibernate to store and retrieve data. Define all the relationships and make sure you use `@OneToMany` and `@OneToOne` relations.

Recommended Readings

1. Thorben Janssen and Steve Ebersole. 2017. *Hibernate Tips: More Than 70 Solutions to Common Hibernate Problems*. Createspace Independent Pub
2. K. Santosh Kumar. 2017. *Spring and Hibernate*. McGraw Hill: New York
3. Hibernate User Guide – https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html
4. Hibernate ORM Guide – <https://hibernate.org/orm/documentation/5.4/>

Consuming Web Services

A.1 | Endpoint Connections



In Chapters 22 and Chapter 23, we built various webservice APIs using Spring and Hibernate. Now, we need to focus on the front end to consume these webservices in order to provide the desired functionality to the users. In Section 24.1, we will focus on the front end to call the webservice APIs.

The following JQuery Ajax code block tries to call Customer web service to fetch customer record by passing email address. This method call is asynchronous in nature which means the processing will take place parallelly and code will not wait for the result to come back from the server. This call uses GET method to pass email parameter to the getCustomerByEmail webservice API. In Chapter 22, we developed a Customer Endpoint “”. We will call this using GET method by passing the customer’s email address.

```
var EmailAddress = "sales@zonopact.com";
// you can get this data from loggedin user account
$.ajax({
  url: "https://localhost/customer/getCustomerByEmail",
  type: "get", //send it through get method
  data: {
    email: EmailAddress
  },
  success: function(response) {
    //On Success, we will get the customer data in a JSON format
  },
  error: function(xhr) {
    //In case of error, we will get the error object and handle this on the front end so
    user knows what is going on in the system
  }
});
```

As you can see from the above example, we are calling the web service endpoint. Please note we are hosting the following web service on our localhost: url: <https://localhost/customer/getCustomerByEmail>

Hence, using the following URL. You can replace this with any URL where you are hosting the web service.

In the success section, we are going to process the response that is in JSON format. Let us take a quick look at how to process the incoming JSON response that contains customer information for the given email address.

The following code shows how to parse the incoming response to get JSON object and then call object properties to get data out of it. In the following example, we are getting customer’s name.

```
var obj = jQuery.parseJSON(response);
alert( obj.name );
This alert will give us customer name. This way you can get other data elements as well.
Finally, this is how the complete Ajax code will look.
var EmailAddress = "sales@zonopact.com"; // you can get this data from loggedin user
account
$.ajax({
    url: "https://localhost/customer/getCustomerByEmail",
    type: "get", //send it through get method
    data: {
        email: EmailAddress
    },
    success: function(response) {
        //On Success, we will get the customer data in a JSON format

        var obj = jQuery.parseJSON(response);
        alert( obj.name );
    },
    error: function(xhr) {
        //In case of error, we will get the error object and handle this on the front end
        // so user knows what is going on in the system
    }
});
```

Create a one-page application which uses JQuery ajax to get weather information from OpenWeatherMap website. You will find the required API on the following site: <https://openweathermap.org/current>

Possible Interview Questions and Answers

B.1 | HyperText Markup Language (HTML)

1. Is it compulsory to close all of the tags in HTML?

Answer: In HTML5, it is not compulsory to close some specific HTML tags. The tags which do not need to have a closing tag are known as *self-closing tags*.

2. What do you understand by “head” in HTML?

Answer: The “head” in an HTML document is the part that is not shown in the web browser whenever the page is fully loaded. It is the first section of the code that comprises of important information about a web page’s links and properties. By exploring an HTML head, you can gather the title of the page, CSS code, meta tags, Open Graph tags along with JavaScript code.

3. How would you use an image map?

Answer: An image map works as a graphical image in which the users can click on different places of the image to be directed to different destinations. Unlike an ordinary image link where the entire image is linked to a single destination. An image map is drafted to hyperlink various areas in the image to various destinations. Image maps are helpful for linking different sections of an image without having the need to create separate image files for an image.

4. Can you create multicolored text on a web page?

Answer: Yes, it is simple to create multicolored text on a webpage. To create text with multiple colors, you need use `....` tags in every character that want to put color. This tag combination can be used as many times as needed.

5. How will you keep list elements straight in an HTML file?

Answer: We can keep the elements straight in an HTML file by using indents for it. The indents specify the amount of white space that is allocated before texts.

6. Can old HTML files work in a new web browser?

Answer: Yes, older HTML files will work in new web browsers as they are compliant to HTML standards. The majority of the old files will continue to work on the newer browsers. However, some features may not work.

7. What do you understand by Applet?

Answer: It is a small application that is used for support and enhancement of hosting web application. Applets are usually created using Java programming language and can run in a web browser.

8. Can you set specific colors for table borders?

Answer: The color of the border can be set with the BORDERCOLOR attribute of the `<TABLE>` tag.

9. If we open a CSS file in a browser, what will be the outcome?

Answer: If we try to open a CSS file in a browser, then the browser will not open the file because the file has a different extension. The only way we can utilize a CSS file is to reference it using <link/> tag within another HTML file.

10. Differentiate between HTML and XHTML?

Answer: HTML and XHTML are mark-up languages in which we write web pages. In comparison, XHTML is more strict in its syntax rules than HTML. HTML is SGML based, whereas XHTML is XML based. We can say they are like two sides of the same coin. XHTML was taken HTML to conform to XML standards. This is the reason why XHTML is strict as compared with HTML. Therefore, we can say that HTML is the dominant mark-up language used for building web pages, whereas XHTML is like a brother of XML languages which is used for extending or mirroring versions of HTML.

11. Is JavaScript supported by HTML?

Answer: Yes, JavaScript is supported by HTML. It is very easy to implement JavaScript. You just need to put your code inside the HTML document and inform the web browser that it is JavaScript. JavaScript can work on web users' computer even if they are not connected to the Internet.

12. How can you use JavaScript with HTML?

Answer: It is very easy to implement JavaScript. You just need to put your code inside the HTML document and inform the web browser that it is JavaScript. JavaScript can work on web users' computer even if they are not connected to the Internet. It allows you to create very responsive interfaces for a web page that greatly improves the user experience and facilitates powerful dynamic functionality.

13. How can you use an iframe?

Answer: An iframe allows us to load separate HTML documents into an existing document. You can place an iframe anywhere in the document flow. The main advantage of the iframe is that it provides the user with the power to decide the size of the portion in which he/she desires to display the content of another webpage.

14. What do you understand by SPAN in HTML?

Answer: The SPAN tag is an inline container used for line content and elements. It has the job to group elements for styling purpose.

15. What do you understand by logical and physical tags in HTML?

Answer: Logical tags are used to explain the meaning of the enclosed text, whereas physical tags are used to specify exactly how particular characters are to be formatted.

16. What is the difference between HTML and HTML5?

Answer: HTML is an older technology, which was developed in 1990. On the other hand, HTML5 is relatively new as it was developed in 2014. The main difference between the two is that audio and video are an integral part of HTML5, while these two elements were not supported in HTML. Before HTML5, people had to rely on Flash as a tool to embed media into web pages. In HTML5, you don't need to rely on third-party applications for embedding audio and video. Vector graphics also got included in HTML5 as standard. In older HTML, vector graphics can be used with the help of different technologies like Silverlight, Flash, VML, etc. In HTML, developers can utilize the web browser cache for temporary storage, while in HTML5 we need to utilize application cache, web storage, SQL database for storing temporary data. HTML is supported by almost every older web browsers. However, the majority of modern web browsers such as Chrome, Firefox, Opera, Edge, Safari, etc. are supporting HTML5.

17. How and why do we use CSS in HTML?

Answer: We use CSS to style the HTML the way we want. There are two methods to use CSS in HTML. We can do it by embedding it right on an index.html file or linking it by creating a separate CSS file.

18. Why do we use Marquee tag in HTML?

Answer: The Marque tag is used in HTML for scrolling a part of text or image that is displayed vertically or horizontally on a web site page.

19. Mention the new structural elements in HTML5.

Answer: There are a number of HTML elements introduced in HTML5. Body, Main, Article, Section, Aside, Nav are of the structural elements of HTML5.

20. Explain the types of headings used in HTML.

Answer: HTML has six levels of headings. Each heading element shows the font changes, paragraph breaks, and any white space required to render the heading. H1, H2, H3, H4, H5, and H6 are the six different heading levels. H1 has the highest importance, while H6 has the lowest.

B.2 | Cascading Style Sheets (CSS)

1. Name the components of a CSS style.

Answer: There are three main components of CSS style – selector, property, value. Selector is an HTML tag on which the style is meant to be applied. Property is a style that is applied to the element. Value is used for assigning the value to a property.

2. What do you understand by a type selector?

Answer: A CSS selector is a part of the CSS rule set that mainly selects the content we want to style. It selects HTML elements according to their id, class, type, attribute, etc. There are various selectors in CSS. These are some examples: CSS Element selector, CSS Id selector, CSS Class selector, CSS Universal selector, and CSS Group selector.

3. What is a block element in CSS?

Answer: There will be Line Break for Block element and we can adjust the height and width of the element.

4. How do we utilize CSS image sprites?

Answer: CSS image sprites are two-dimensional images which are built of combining small images into one single larger image with defined X and Y coordinates. To execute this, you can use CSS background-position properly by defining the exact placement of the image to be shown.

5. How and when CSS originated.

Answer: On 10 October, 1994, Hakon Wium Lie first proposed the concept of CSS.

6. Mention the limitations of CSS.

Answer: Although CSS is very powerful for designing, it cannot perform any arithmetical and logical task. More limitations:

- CSS cannot interact with databases
- It cannot request a web page
- We cannot read files by using CSS

7. Differentiate between inline, embedded, and external style sheets.

Answer: With inline CSS attribute, you can style attribute of any HTML element to set the style rules. With Embedded CSS you can define your CSS rules into HTML document by using the <style> element. With External CSS, you can include an external style sheet file in your HTML document.

8. Which property would you use to define the width of the outline?

Answer: An outline is a line that is drawn near the elements to make the element show well. We use the outline-width to specify with the width of an outline.

9. Which syntax is used to add multiple background images in CSS3?

Answer: The CSS Multi background property is used to adding multiple images at a time without HTML code. By this, we can add images according to our requirements. An example of the syntax of multibackground images:

```
#multibackground {
background-image: url(/css/images/logo.png), url(/css/images/border.png);
background-position: left top, left top;
background-repeat: no-repeat, repeat;
padding: 75px;
}
```

10. How would you use the transition effect in CSS3?

Answer: We use the transition property in CSS to make some transition effects. This effect is a combination of four properties; transition property, transition duration, transition timing function, transition delay.

B.3 | JQuery

1. Explain where JQuery is used?

Answer: JQuery works as a JavaScript library. It is used for HTML DOM access and it facilitates the most important feature of event handling as well.

2. Differentiate between JavaScript and JQuery.

Answer: JQuery is a library, while JavaScript is a language. JQuery is responsible for providing fully-functional support for JavaScript language.

3. What is the important of “\$” sign in JQuery?

Answer: The “\$” is used as a designation for JQuery. Example: \$('#MyControl')

4. How will you hide and show controls in JQuery?

Answer: To hide and show the controls you need to access the control using “\$” sign and utilize the methods Hide() and Show().

5. What do you understand by selectors in JQuery?

Answer: A JQuery selector is a functionality which utilizes expressions to locate matching elements from a DOM based on certain criteria. They are used to select HTML elements using JQuery.

6. Differentiate between “length” and “size” in JQuery.

Answer: Both “length” and “size” are used for finding the number of elements in an object. We use “length” commonly because it is faster as compared with “size” because “length” stands a property, while “size” is a method.

7. Why do we use JQuery connect?

Answer: We use JQuery for binding one function to another. It is used for executing the function when a function is executed from another object.

8. How do we use AJAX in JQuery?

Answer: Ajax will be implemented differently by different browsers. This means if you are adopting the normal JavaScript way to implement the Ajax, then you will need to write the different code for various browsers to ensure that Ajax would work cross-browser.

9. What do you understand by Content Delivery Network? Mention its advantages.

Answer: Content Delivery Network (CDN) is a system of computers that are present all over the world to cache files for users to access. CDNs can immensely reduce the load time of web page by delivering files at a faster and higher bandwidth from a server that is closer to your visitor than your own server might be.

10. Explain the `empty()` method in JQuery.

Answer: The `empty()` method is responsible for removing all child nodes and content from the selected elements. However, to remove the elements without removing elements and data, we use the `detach()` method.

11. How would you use the `clone()` method in JQuery?

Answer: The `clone()` method is responsible for cloning matched DOM Elements and selects the clones. You can use this useful method for moving copies of the elements to another location in the DOM.

12. How would you use the validation JQuery plugins?

Answer: Using a JQuery plugin is a simple process. It includes:

- Include JQuery: You need to include JQuery v1.x as the validation plugin.
- Include the JQuery Validation Plugin.
- Create the HTML form.
- Create styles for the form.
- Create the validation rules.

13. Is it possible to include different versions of JQuery in a web page?

Answer: Yes, it is possible to use multiple versions of JQuery on the same page. You will need to use the `JQuery.noConflict()` method to avoid any kind of conflict. This method is useful for using multiple frameworks while using JQuery.

14. Explain caching in JQuery.

Answer: The JQuery caching is focused on the data function. Like any other JQuery functionality, data also applies to the wrapped set resulting from a query. JQuery caching is implemented as a plain dictionary in which an individual element is characterized by a value and a name.

15. Explain the `slideToggle()` method in JQuery.

Answer: The `slideToggle()` method is used for toggling between `slideUp` and `slideDown` for the selected elements. This method is useful for selecting the elements of visibility.

B.4 | Bootstrap

1. Mention the different ways of adding Bootstrap to a web project.

Answer: You can simply add Bootstrap into a web project by using the `<link>` and `<script>` tags in the header of the HTML file. We use the `<link>` tag for including Bootstrap style sheet in the project whereas, the `<script>` tag is utilized to add the JavaScript files that come along with Bootstrap.

2. Name the latest version of the bootstrap.

Answer: Currently, web developers are using Bootstrap 4.1, which is the latest stable version of Bootstrap.

3. Give one reason for choosing bootstrap over regular CSS files.

Answer: Bootstrap works as a CSS framework. With Bootstrap we can develop our websites quickly without needing to write the basic building blocks from zero all over again. This functionality saves a lot of time and lets us focus more on business logic. It also delivers inbuilt styles, standard components, and CSS classes. Bootstrap also helps in building fully responsive websites. These websites automatically adjust themselves according to the screen size to look equally good on every single device.

4. Can you create a simple button by using the classes in Bootstrap 4?

Answer: To do this we need to use the following command: <button class="btn btn-primary">This is a Button</button>

5. How did Bootstrap originate?

Answer: Mark Otto and Jacob Thornton at Twitter developed Bootstrap as a framework to encourage uniformity across internal tools. Originally, Bootstrap was named Twitter Blueprint and before it, different libraries were used for the development of the interface. This led to the inconsistencies and a high maintenance burden.

6. What do you understand by space utilities in Bootstrap 4?

Answer: Developers use padding spaces and margins to regulate how components and elements are spaced and sized. You will find a five-level scale for spacing utilities, based on a 1rem value default \$spacer variable.

7. What is a “card” in Bootstrap?

Answer: In Bootstrap 4, a card is a bordered box with padding near its content. This card will show you options for headers, footers, content, colors, etc.

8. What is the use of carousel in Bootstrap?

Answer: A carousel in Bootstrap gives you the functionality of slideshow to cycle through a series of content. They are built with CSS 3D transforms and some JavaScript. It could work well with multiple images, text, or your own custom mark-up.

9. What are “breadcrumbs” in Bootstrap?

Answer: Breadcrumbs in Bootstrap shows us the navigational hierarchy of a website. The separators in this hierarchy are automatically added by using CSS. We can also use breadcrumbs for the Documentation page, Magazine, and ERP system.

10. What do you understand by “badges” in Bootstrap?

Answer: Badges in Bootstrap are used for adding additional information to any content. You can use the .badge class together with another contextual with elements to develop rectangular badges.

B.5 | Java

1. Differentiate between final, finally, and finalize in Java.

Answer: Final works as a modifier by which you can apply to methods, classes, variables. If you build a variable final, this means its value cannot be varied once built. Finalize is a method which is executed just before an object is garbage collected, permitting it the last chance to save itself, but this call to finalize is not definite. Finally is a keyword that is used for exception handling, with try and catch. The finally block is always implemented regardless of whether an exclusion is thrown from try block or not.

2. Mention the list of Object class methods.

Answer: There are various object class methods:

- **Clone():** It creates and gives a copy of an object.
- **Equals():** This indicates whether an object is “equal to” this particular one.

- **`Finalize()`:** This is called by the garbage collector or an object whenever the garbage collection decides that there are no more mentions to the object.
- **`getClass()`:** It returns the runtime class of an object.
- **`hashCode()`:** It returns a hash code value for the object.
- **`notify()`:** It wakes up a single thread which is waiting on this object's monitor.
- **`notifyAll()`:** It wakes up all threads that are waiting on this object's monitor.
- **`toString()`:** It returns a string depiction of the object.
- **`wait()`:** It causes the current thread to wait until another thread summons the `notify()` method or the `notifyAll()` method for this object.

3. What are the advantages of using Java?

Answer: Java is one of the most famous programming languages in the world. It is easy to learn. Java was developed to be easy to use and thus easy to write, debug, compile, and learn the other languages for programming. Java is object oriented; this means it allows you to create reusable code and modular programs. One of the most significant advantages of Java is platform-independence. It could easily move from one computer system to another. Java's robustness, ease of use, cross-platform capabilities and its security features make it the number one choice for providing worldwide Internet solutions.

4. What is the Object and Class in Java?

Answer: Object and Class are two basic concepts commonly used in object-oriented programming (OOP). A class works as a user-defined design or concept from which objects are created. An object is a basic unit of OOP and shows the real-life entities.

5. What do you understand by JVM, JDK, and JRE?

Answer: Java virtual machine (JVM) is an abstract machine. It works as a specification to deliver a runtime environment in which Java bytecode can be executed. JRE stands for Java Runtime Environment. It is the real physical implementation of JVM. It consists set of libraries with other files that JVS uses. JDK stands for Java Development Kit. It consists of JRE and development tools.

6. Differentiate between Overloading and Overriding.

Answer: Overriding and Overloading are two important concepts in Java. They might create confusion for Java novice programmers. Overloading takes place when two or more methods in one class have the identical method name but different parameters. Whereas Overriding means having two methods having the same method name and parameters.

7. What is meant by inheritance?

Answer: Inheritance is a system in which new class is derived from an existing class. In Java, classes can inherit or gather the properties and methods of the other classes. A class that is derived from another class is called subclass. On the other hand, the class from which a subclass is derived is known as superclass.

8. What is the use of ClassLoader?

Answer: Java ClassLoaders are a very important part of Java Virtual Machine (JVM). It is used for loading a Java class into JVM.

9. What is object-oriented paradigm?

Answer: Object-oriented programming is a programming paradigm based on objects that aim to incorporate the advantages of reusability and modularity.

10. What is WORA?

Answer: WORA stands for Write Once Run Anywhere. It is a feature applicable to those programs which have the power to execute itself on a different operating system or any computer or machine.

B.6 | Spring

1. What is Spring Core?

Answer: Spring has been developed in such a way that Spring is broken down into different parts known as modules. Numerous modules are present within the Spring framework. In these, one module is known as Spring Core. It is useful for the management of the dependencies and managing complexity that is there with the components.

2. Why is Spring considered to be one of the most popular Java related frameworks?

Answer: Spring Framework provides the development of loosely coupled classes based on well-designed interfaces. This makes writing testable code easier. The Spring framework provides great integration support with other non-Spring frameworks.

3. Mention the major features in different versions of Spring.

Answer: Spring 2.5 introduced annotation-driven configuration. Spring 3.0 introduced the great of Java 5 improvements in language. Spring 4.0 is the first version to introduce Java 8 features.

4. List some of the latest specifications supported by Spring 4.0.

Answer: Spring 4.0 is very useful for supporting Java EE7 specifications. It supports JMS 2.0, JTA 1.2, JPA 2.1, Bean Validation 1.1, and JSR-236 for concurrency.

5. How is the process of validation executed by using the Spring Framework?

Answer: Spring validator can be utilized in both web and business layers to validate objects. It is based on the org.springframework.validation. We can use the following methods:

- **Supports(class):** This validator supports a specific class.
- **Validate:** This validates and sets errors into Errors objects.

6. What is the difference between @Component, @Service, @Repository and @Controller?

Answer: Typical Java backend web services or applications are based on MVC (Model, View and Controller) pattern. Hence, the application is developed in multiple layers in which Controller is the main communication point for external applications. As you know Spring is based on dependency injection and inversion of control design patterns and hence it is important for Spring to know which POJO (Plain Old Java Object) class is intended for which purpose. For this, we use annotations on the classes to identify their roles. In order for telling Spring about our Controller classes, we use @Controller annotation. Similarly, the business logic remains in the service classes which communicate with models to read and write data from the database. These classes are denoted by @Service annotation. Now, as you might have guessed, our model classes are denoted by @Repository annotation. All three annotations @Service, @Repository and @Controller are special types of @Component annotation which you have seen used for specific purposes. Hence @Component can be used for any layer.

7. What are the various types of scopes of Bean?

Answer: In Spring, <bean> definition allows to mention a scope. There are five types of scopes supported by Spring. If no scope is provided, Spring uses “singleton” as a default scope. See the following explanation of all the scopes.

Singleton: If this scope is used on the bean definition or no scope is used, Spring creates only one instance of the bean per IoC container. It means, each time we ask for a bean instance, the same instance is provided by Spring.

Prototype: If this scope is used on the bean definition, each time we ask for a bean instance, Spring creates a new instance.

Request: This scope is only for web-aware ApplicationContext. If this scope is used on the bean definition, upon each HTTP request Spring creates a new instance.

Session: This scope is only for web-aware ApplicationContext. If this scope is used on the bean definition, Spring creates a new instance on each HTTP session.

Global-Session: This scope is only for Portlet applications. Each portlet has its own session, but if you want to have a bean for all the sessions in portlets you can use Global-Session on the bean definition. In this case, Spring creates a new instance on each global HTTP session.

8. What are the benefits of using Spring Framework?

Answer: Spring is a very powerful framework. It addresses many problems faced in Java EE. This includes support for the management of business objects. It provides great programming practice like programming utilizing interfaces instead of just classes. Spring makes it possible for the developer to develop enterprise applications. It is also modular which allows you to use only those parts that are needed. Plus, it supports XML and annotation-based configuration. Lastly, Spring Test module facilitates support for the simple easy-to-test code.

9. What do you understand by ApplicationContext in Spring?

Answer: The ApplicationContext interface works as a central interface within a spring application that is utilized for delivering configuration information to the application. It is known for implementing the BeanFactory interface. Therefore, its main job is to support the development of big business applications.

10. Differentiate between Beanfactory and ApplicationContext in Spring.

Answer: There are numerous differences between BeanFactory and ApplicationContext:

- BeanFactory never provides support for the internationalization, whereas ApplicationContext can provide support for this purpose.
- Both BeanFactory and ApplicationContext has a different ability to publish an event to beans that are recorded as a listener.
- On the one hand, BeanFactory uses lazy initialization approach. On the other hand, the Application context uses eager initialization.
- BeanFactory supports Annotation-based dependency injection, whereas BeanFactory ApplicationContext supports @PreDestroy or @Autowired.

B.7 | Hibernate

1. What do you understand by Hibernate?

Answer: Hibernate is an object-relational mapping framework. It provides the developer with the capability to focus on business logic by taking care of the persistence of data by its own. A Java developer can easily write code by using an object. Then, Hibernate can do the job of creating those objects from data loaded from the database and uploading the data again to the database.

2. What is meant by ORM?

Answer: ORM stands for object-relational mapping. It works as the fundamental for Hibernate framework which is responsible for mapping database tables with Java Objects. It facilitates various API's to perform various types of operations on those data tables.

3. Mention the advantages of Hibernate over JDBC.

Answer: The greatest advantage of using Hibernate is persistence. This means the saving and loading data from Database. It also facilitates benefits to caching, lazy loading, relationship management by providing code for mapping an object into the data. This makes the developer free from writing lines of code.

4. Name the different types of caches available in Hibernate.

Answer: Hibernate delivers innovative methods of caching solutions. This includes first level caching, second level caching, and query cache.

5. Is SessionFactory thread-safe in Hibernate?

Answer: No, Session object will not be thread-safe in Hibernate. It is intended to be used within single thread in the application.

6. Differentiate between session and SessionFactory in Hibernate.

Answer: A session is a single-threaded and short-lived object, whereas SessionFactory is immutable and shared by all session. It also keeps running until the Hibernate is running.

7. What do you understand by Hibernate Query Language?

Answer: Hibernate Query Language, or HQL, works as an object-oriented extension to SQL. This makes it possible to query, store, update, and retrieve objects from a database without having the need to use SQL.

8. Differentiate between sorted and ordered collection in Hibernate.

Answer: Sorted collection collects and sorts the data in JVM's heap memory using Java's collection framework. Whereas, the ordered collection is sorted out by using order by the clause in the database itself.

9. Mention the two types of collections in hibernate.

Answer: There are two types of collection in Hibernate. First is Sorted collection and the second is Ordered collection.

10. How will you get Hibernate statistics?

Answer: You can obtain statistics by using `getStatistics()` method of SessionFactory class by this: `SessionFactory.getStatistics()`

B.8 | Model-View-Controller (MVC)

1. Mention the advantages of MVC.

Answer: MVC supports rapid and parallel development. By this, a programmer can work on the business process of the web application. You can also create multiple views of a model. It supports asynchronous technique for the developers to develop an application that loads very fast. Plus, if the developer makes a modification, then it won't affect the entire model as model parts do not depend on the views part. Lastly, MVC is also a very SEO friendly platform for generating SEO-friendly URLs.

2. Do we face ViewState in MVC?

Answer: No, MVC does not have any ViewState. It is due to the fact that ViewState is stored in a hidden field on the page.

3. What do you understand by Routing in MVC?

Answer: Routing is the method of directing an HTTP request to a controller and the utilization of this processing is implemented in System. MVC framework uses system web routing for directing a request to a controller.

4. What is meant by Output Caching in MVC?

Answer: Output Caching lets you save the response of the action method for a specific duration. This greatly assists in performance as the saved response gets returned from the action method and does not create a new response.

5. What is a View Engine?

Answer: View Engines are very useful for creating the HTML from the views. Views consist of HTML and source code in few programming languages like C. View Engines develops HTML from the view which is then returned to the web browser and rendered. Examples of View Engines are WebForms and Razor.

6. Elaborate BundleConfig.cs in MVC4.

Answer: BundleConfig.cs in MVC is utilized for registering the bundles by the bundling process and minification system. Various bundles are added by the default jQuery libraries such as jquery.validate, Modernizr, etc.

7. Name the important namespaces used in MVC.

Answer: These are some important namespaces that are used in MVC:

- System.Web.Mvc
- System.Web.Mvc.Ajax
- System.Web.MVC.html
- System.Web.MVC.Async

8. Differentiate between ViewBag and ViewData in MVC.

Answer: ViewBag works like a wrapper that wraps around the ViewData. It makes it possible to create dynamic properties. There's no need to typecast the objects as in ViewData. However, ViewBag is slower than the ViewData.

9. What is understood by HTML helpers in MVC?

Answer: HTML helpers work like controls in the traditional web forms. However, these HTML helpers are much lighter compared to the web controls because it does not hold ViewState and the events. HTML helpers are responsible for returning the HTML string which can directly be rendered to an HTML page.

10. What is meant by "layout" in MVC?

Answer: We use layouts in MVC to get a consistent look and feel on every page of our application. It is very similar to providing master pages. However, MVC gives us more functionalities.

B.9 | REST

1. What is known as safe REST operations?

Answer: The REST API utilizes HTTP methods to execute its operations. Some of these HTTP operations which does not change the resource at the server are known as safe operations. Example: GET and HEAD.

2. Mention the advantages of using the REST template.

Answer: The REST template is very useful for the implementation of method pattern in Spring framework. It simplifies the interaction with RESTful Web Services for the client side. We can use it for consuming a RESTful Web Service very easily.

3. Which HTTP methods are used by REST?

Answer: These are the HTTP methods supported by REST:

- **GET:** It is used for requesting a resource at the request URL.
- **POST:** It is used for submitting information to the service for processing.
- **PUT:** By the request URL it updates the resource.
- **DELETE:** By the request URL it eliminates the resource.
- **OPTIONS:** This shows which techniques are being supported.
- **HEAD:** It shows about the request URL and it returns meta information.

4. What do you understand by RESTful web service?

Answer: The web services that are developed by using the REST style are called RESTful web services. These web services are known for using HTTP methods to adopt the concept of the REST architecture. Such web service usually defines a URI and delivers resource representation like JSON.

5. Name the protocol used by RESTful web service.

Answer: The HTTP protocol is used by the RESTful web services to make communication between the client and the server.

6. Explain payload in RESTful web service.

Answer: The request body present in every single HTTP message includes request data known as the payload. This specific part of a message is the main interesting part for the recipient.

7. Draw a comparison between SOAP and REST.

Answer: SOAP works as a protocol that makes it possible for two computers to communicate with each other by sharing XML documents. Whereas REST works as a service design and architecture meant for network-based software architecture. SOAP only supports XML format, while REST can support various data formats. SOAP doesn't have caching feature, while REST can support caching. REST remains faster than SOAP.

8. Which Java API helps in the development of a RESTful web service?

Answer: There are numerous frameworks and libraries that can be used for creating RESTful web services in Java. Frameworks and libraries like JAX-RS, RESTEasy, RESTlet, and Apache CFX are some useful tools for creating RESTful web services.

9. Tells us about the resources present in a REST architecture.

Answer: The REST architecture is known for treating every single content as a resource. These identified resources can be text files, HTML pages, videos, images, or simply business data. A REST server facilitates access to resources and REST client accesses and changes these resources.

10. Elaborate on the key characteristics of REST.

Answer: REST gives us scalability, performance, simplicity, portability, and modifiability. Along with this, the REST API makes it possible for the different systems to communicate in a very simple way. Every single REST API call will have a relation between an HTTP verb and the URL. REST is a stateless but cacheable architecture that is not a protocol but a pattern.

B.10 | Web Services

1. What is understood by web services?

Answer: A web service is a software that is accessible over the Internet. It uses the XML messaging system and delivers easy to use and understand user-interface for the end users.

2. Differentiate between SOA and a web service.

Answer: The SOA is an architecture and design meant to implement other services. It can easily be implemented by using various protocols like HTTP, HTTPS, SMTP, RMI, IIOP, etc. Whereas a web service itself is applied technology. You can implement SOA by using a web service.

3. Give an example of real web service.

Answer: The IBM web services browser is a real example of a web service.

4. Define web service protocol stack.

Answer: Web service protocol stack is a stack of various protocols that can be utilized for exploring and executing web services. The entire stack consists of four different layers: Service Transport, XML Messaging, Service Description, and Service Discovery.

5. Explain Web Services Description Language.

Answer: Web Services Description Language (WSDL) is used for describing a web service. It helps in specifying the location of the service, and the methods of the service. It is basically an XML based file which tells the client application what the web service does.

6. Mention the security measures needed for web services.

Answer: The security measures taken for web services must be more than that of what we say Secure Socket Layer (SSL). We can achieve this level of security by implementing the Entrust Secure Transaction Platform. This level of security is needed in web services for ensuring reliable transactions and safe confidential information.

7. Mention the tools used for testing a web service.

Answer: We can use SoapUI for SOAP WS and Firefox poster plugin for RESTful Services.

8. Do we need any specific application to use web service?

Answer: No, we do not have any specific application to use a web service. The name itself is self-explanatory, we can access a web service from any application that supports XML based object request and response.

9. Name some opensource and commercial implementations of web service.

Answer: Apache SOAP, REST, JAX-WS Reference implementation, Oracle Java EE Metro, Apache CXF are some examples of web services implementations.

10. Which browser allows access to web service?

Answer: JavaScript XMLHttpRequest object is important to use web service through a web browser. Internet Explorer, Safari, Chrome and Firefox support web services.



Answers to Objective Type Questions

Chapter 1: Introduction to Full Stack Development

1. (a) getCurrentPosition ()
2. (d) window.captureEvents (Event.CLICK);
3. (d) Default Status
4. (b) Semicolon, colon
5. (a) True

Chapter 2: Getting Started with Full Stack Development: A Project Idea

1. (d) Multi-value attributes
2. (c) Flowchart
3. (d) None of the above
4. (a) Spot potential problems
5. (d) Dotted Oval

Chapter 3: Introduction to Hyper Text Markup Language

1. (a) True
2. (d) <summary>
3. (b) Canvas
4. (d) crossOrigin
5. (d) All of the above

Chapter 4: Introduction to Cascading Style Sheets

1. (a) text-align
2. (a) Accelerator
3. (d) All of the above
4. (a) Selector
5. (d) CurrentColor Keyword

Chapter 5: Introduction to jQuery

1. (c) preventDefault()
2. (a) True
3. (a) Sort()
4. (b) empty()
5. (b) filter (selector)

Chapter 6: Introduction to Bootstrap

1. (b) 1.428
2. (b) Providing different icons
3. (b) Fluid layout
4. (c) .img-thumbnail
5. (d) modal

Chapter 7: Build Pages for MyEShop with HTML and CSS

1. (b) <p>
2. (a) How to display the page.
3. (c) CGI-BIN
4. (d) border-bottom-left-radius
5. (b) border-collapse

Chapter 8: Use of jQuery on HTML CSS

1. (a) .toggle()
2. (b) data()
3. (b) False
4. (b) hide()
5. (b) JavaScript

Chapter 9: Use of Bootstrap to Make HTML Responsive

1. (a) 12
2. (b) Carousel
3. (c) 20px
4. (a) list-group
5. (c) .danger

Chapter 10: Introduction to Java Language

1. (d) Javac
2. (d) Java
3. (a) Javadoc
4. (a) Java Archive
5. (d) Finalize

Chapter 11: Language Syntax and Elements of Language

1. (c) 64
2. (d) /*
3. (b) char
4. (a) True
5. (b) '23'

Chapter 12: Object-Oriented Programming

1. (d) Explicit
2. (a) Multitasking
3. (b) Initializing a newly created object
4. (a) Overriding
5. (a) Object-Oriented Programming System

Chapter 13: Generics and Collections

1. (d) Generics are useful for adding stability to the code by making bugs detectable at compile time.
2. (b) N
3. (b) Type Interface
4. (b) Java.util
5. (c) A group of objects

Chapter 14: Error Handling

1. (b) Run time
2. (b) thrown
3. (b) throw
4. (c) Throwable
5. (d) None of the above

Chapter 15: Garbage Collection

1. (b) finalize()
2. (b) JVM
3. (b) Young Space
4. (d) Mark and sweep model
5. (b) JVM

Chapter 16: Strings, I/O Operations, and File Management

1. (b) String class is defined in java.util package.
2. (d) equals()
3. (b) FileReader
4. (a) read()
5. (b) write()

Chapter 17: Data Structure and Integration in Program

1. (a) Linear Data Structure
2. (b) Unsigned
3. (a) Two
4. (a) True
5. (c) Expression Tree

Chapter 18: Lambdas and Functional Programming

1. (b) False
2. (a) Predicate
3. (a) True
4. (b) LocalTime.now();
5. (a) Yes

Chapter 19: Multithreading and Reactive Programming

1. (c) Process and thread based
2. (b) Integer
3. (d) start()
4. (b) 5
5. (b) False

Chapter 20: Introduction to Spring and Spring MVC

1. (a) DispatcherServlet
2. (d) RequestMapping
3. (c) Validator
4. (a) Yes
5. (b) HandlerAdapter

Chapter 21: Introduction to Hibernate

1. (b) SessionFactory
2. (d) Dselect fetching
3. (c) Query By Criteria
4. (c) .hbm
5. (c) Third Level

Chapter 22: Develop Web Services for the APIs

1. (c) @Autowired
2. (a) Model View Controller
3. (b) Container for classes
4. (a) Provide access to an underlying database

Chapter 23: Develop Models with Hibernate

1. (a) Relational Database Management System
2. (c) CascadeType.SAVE
3. (b) False
4. (a) True
5. (a) @GeneratedValue

Index

A

abstraction, 341–342
 benefits of, 342–343
 real-world understanding, 342
 use of abstract classes, 342
abstraction341-342
adjacency list, 455
adjacency matrix, 454–455
admin dashboard flowchart, 29–30
amazon.com, 1
anonymous objects, 421–422
API-based architecture with REST, 12–13
application tier, 2
ArrayList, 386–388
artificial intelligence, 350
autoboxing process, 311–313

B

back-end application, 1–2
back-end development with Java 11, 11–12
back-end technologies (server-side), 11
back-end web services API endpoints, 32–34
Berners-Lee, Sir Tim, 5
Binary search tree, 451
Binary Tree, 450
Bootstrap, 8–10
 alerts, 191–196
 buttons, 196–201
 cards, 208–209
 carousel, 219–222
 colors, 185–187
 display headings, 182–183
 forms, 217–219
 grid system, 179–181
 home page, 253–256
 HTML's elements in, 183–184
 images, 187–189
 installation, 177
 jumbotron, 190–191
 media objects, 223–227
 navigation bar, 213–217
 navigation menus, 209–213
 pagination, 206–208
 prerequisites, 177
 progress bars, 201–205
 setting up environment, 251–256
 structure of webpage, 178–179

 typography, 182–184

bootstrap.js, 251
branching statements, 331–333
breadth-first traversal, 451

C

C#, 1, 11
cart finalization flowchart, 26–27
Cascading Style Sheets (CSS), 4
 backgrounds, 82–85
 box model, 87–88
 “*class*” attribute, 76–77
 colors, 80–82
 comments, 77–78
 dropdowns, 111–113
 fonts, 94–95
 forms, 113–114
 group of selectors, 77
 “*id*” attribute, 75
 integration with HTML, 78–80
 links in, 95–97
 lists in, 98–99
 navigation bars, 108–110
 outline, 88–91
 overview, 73
 positioning in, 104–107
 relationship between HTML and, 73–74
 setting up height/width, 85–87
 syntax, 74–75
 tables in, 100–104
 text in, 91–94
 working of, 74
client tier, 2
compiler, Java, 263
compound expressions, 322
concatenation, 428–430
concurrency, 494–504
 designing concurrent Java program, 511–514
concurrent data structures, 506
controller classes, creating, 628–634
cross-site scripting (XSS) attack, 252
customer relationship management (CRM), 19

D

data access object (DAO)
 creating, 618–628
 for CRUD operations, 646–649

database and tables, creating, 638–646
 database tier, 2
 data structures, 441
 non-primitive, 444–457
 primitive, 442–444
 Date and Time API, 474–481
 deadlock, 504–506
 depth-first traversal, 451
 directed graph, 453–454, 456–457
 dynamic polymorphism, 348

E

Eclipse, 291–292
 advantages, 292
 class behaviors, 296–297
 creating packages, 294
 creating projects, 294
 defining classes, 294–296
 environment, 292
 IDE view, 293
 setting up, 292–297
 testing in, 296
 e-commerce
 advantages, 19–20
 considerations while developing, 20–21
 definition, 19
 entity relationship diagram of, 23
 required entities, 21–22
 UML class diagram, 24
 encapsulation, 339–340
 advantages of, 341
 ENQUIRE, 5
 entity relationship diagram (ERD), 22, 231–232
 error handling
 checked exceptions, 412–413
 importance of, 409–412
 logical errors, 404–406
 runtime exceptions, 413
 semantic errors, 408–409
 syntactical errors, 406–407
 understanding, 403–404
 event handler content attributes, 42–44
 required for document objects only, 45
 with specific rule, 44–45
 Expression Tree, 451–452
 Extensible Markup Language (XML), 15
 external style sheets, 78–79

F

flowchart
 arrow, 24
 diamond, 25
 oval, 22
 rectangle, 24
 rectangle with three sections, 25

Forest, 449–450
 “for” loop statement, 330–331
 front-end application, 1–2
 front-end page flow diagram, 31
 front-end technologies, 3
 full stack web development
 definition of, 1–2
 technologies essential for, 2
 functional programming, 459
 evaluation, 461–462
 higher-order and first-class functions, 460
 in Java, 462–464
 object-oriented *vs.*, 464–465
 pure functions, 460–461
 recursion techniques, 461
 referential transparency, 462

G

garbage collection, 415–416
 advantages, 418
 Concurrent Mark Sweep (CMS), 417
 Epsilon, 422–423
 Garbage First (G1), 417
 major, 416–417
 objects eligible for, 418–422
 garbage collector, 265
 Garbage First Garbage Collector (G1GC), 290–291
 generic programming, 363–364
 application in Java, 365–366
 benefits, 364–365
 `equals()` method, overriding, 369–371
 generic method, 366–367
 `hashCode()` method, overriding, 371–373
 `toString()` method, overriding, 368–369
 GET endpoints, 32–34
 global attributes of HTML5, 40–42
 graphical user interface (GUI), 303

H

HashMap, 378–380
 HashSet, 383–384
 Hashtable, 380–381
 Hibernate, 15–16, 565–566
 architecture, 566–568
 cache, 568
 caching, 592–594
 collection mapping, 579–583
 configuration object, 567
 Criteria query, 567
 inheritance mapping techniques in, 572–578
 installation and configuration, 568–570
 Java objects in, 571–572
 mapping with map, 583–588
 named query, 590–592
 query language, 588–592

- session, 567
 Spring integration, 594–597
 Hibernate Criteria Query Language (HCQL), 590
- HTML pages
 adding CSS to, 238–240
 category list, 235
 execution of code, 236–238
 getting started, 231–238
 header, 234
 home page, 233–234
 logo, 234
 navigation menu, 236
 search bar, 235
 search button, 235
 shopping cart, 235–236
 table, 234
- human–computer interaction (HCI), 3
- HyperText Markup Language (HTML), 1, 4
 abbreviation, *abbr* tag, 56
 alt attribute, 61–62
 attributes, 39–42
 attributes to style elements, 66–69
 bold text, *b* tag, 50
 bookmark, 59–60
 closing tag, 69
 code structure, 37
 comments, 57
 contact information, *address* tag, 56
 different screen sizes, 68–69
strong element, 50
 element hierarchy, 38
 elements, 39
 emphasized text, *em* tag, 51
 event handlers, 42–45
 fonts, 67
 format of tags, 39
 getting started, 38
 headings, 45–47
 images, 61–62
 images as links, 58–59
 inserted text, *ins* tag, 53
 line breaks, *br* tag, 49
 links, 57–60
 lists, 64–66
 marked text, *mark* tag, 52
 paragraphs, *p*, 49
 quotation from other sources, *blockquote* tag, 55
 removed text, *del* tag, 53
 short quotations, *q* tag, 54–55
 size of text, 68
 small text, *small* tag, 52
 style attribute, 66–67
 subscribed text, *sub* tag, 54
 superscripted text, *sup* tag, 54
 tables, 62–63
 head Tag, 47–48
- meta* tag, 47–48
script tag, 48
style tag, 47
title tag, 47
 target attribute, 58
 text alignment, 68
 text color, 67
- HyperText Transfer Protocol (HTTP), 1
- I**
- identifier, 300
 incidence matrices, 456–457
 inheritance, 343–344
 advantages of, 345
 inline style, 79–80
 Integrated Development Environment (IDE), 262, 599
 internal style sheets, 79
 iPhone X, Bootstrap page on, 253–254
- J**
- Java, 1
 arithmetic operators, 318–319
 arrays, 305–307, 446–447
 assignment operators, 314–316
 autoboxing and unboxing process, 311–313
 basic concepts, 262–265
 bitwise and bit shift operators, 324–325
 braces , 300
 branching statements, 331–333
 building blocks of, 299–303
 capitalization, 406
 code blocks, 302–303
 comments, 303
 compiler, 263, 300, 312
 concat () method, 430
 control flow, 327–329
 curly braces, 407
 data types, 301
 developing logic, 313–325
 development environment, 599–601
 Development Kit (JDK), 264
 enums, 307–309
 equality and relational operators, 320–321
 expressions, 326–327
 extension of .java, 300
 file management in, 438–439
 “for” loop statement, 330–331
 garbage collector, 265
 “if–else–if” conditional statements, 328
 “if” statement, 327
 instanceof operator, 321–322
 intValue() method, 313
 I/O, 436–438
 keywords, 300–301
 literals, 302

- logical operators, 322–324
- loops, 329–331
- `main()` method, 300, 303
- object assignments, 316–318
- objects in, 265, 350–351
- primitive assignment, 314
- primitive casting, 314–316
- primitive wrapper classes, 301
- `println()` method, 300
- programming in, 270–279
- reference variable, 316
- role of strings in, 425–428
- Runtime Environment (JRE), 263
- semicolon sign “;”, 300
- splitting strings, 406–407
- static and non-static methods, 303–304
- `StringBuilder` class and `StringBuffer` class, 433–436
- string class, 407
- string operations in, 428–433
- string options, 304–305
- “switch” statement, 328
- ternary operator, 324
- unary operators, 319–320
- values for primitives, 301
- variables, 302
- Virtual Machine (JVM), 259, 263
- “while” loop statement, 329–330
- wrapper classes, 309–311
- Java 9
 - API process, 291
 - Garbage First Garbage Collector (G1GC), 290–291
 - hash algorithms, 288
 - jshell, 287
 - linking process, 288
 - modular packages, 288–290
 - multi-release JAR files, 287
 - platform support, 287–288
 - tuning improvements, 290
 - XML Catalog, 291
- Java collections, 394
 - application, 373
 - arrays, 393–394
 - classes, 377–378
 - comparable interface, 395–396
 - Comparator interface, 397–400
 - difference between Comparable and Comparator interfaces, 400
 - interfaces, 375–377
 - key methods, 394–395
 - linked list implementation, 374
 - list interface, 386–390
 - map interface, 378–383
 - object collections, 373–378
 - queue interface, 390–393
 - set interface, 383–386
- stack library in, 374
- types of, 374
- vector class, 388–389
- Java Development Kit (JDK), 599
- Java packages
 - constructors, 281–284
 - default constructor, 281
 - difference between constructors and methods, 284
 - instance methods, 285
 - methods, 281
 - programming examples, 285–286
 - static methods, 285
 - structure, 279–280
 - “super” keyword, 283–284
 - “this” keyword, 283
 - variables, 280–281
- Java Runtime Environment (JRE), 300
- JavaScript, 6
- JavaScript Object Notation (JSON), 14–15
- Java 13 version, 259–261
 - features, 259–262
- Java virtual machine (JVM), 11, 299
- JDK Enhancement Proposals (JEPs), 259
- jQuery, 7
 - `addclass()`, 153–154
 - `after()` and `before()` method, 148–149
 - ancestors, 158–159
 - animation in, 132–135
 - `append()` method, 144–146
 - callback function, 136–137
 - chaining, 137–138
 - changing and manipulating HTML elements and attributes, 138–153
 - `children()` method, 162–163
 - `class` selector, 121–122
 - `click()` function, 122–123
 - configuration of, 117–118
 - with CSS, 153–158
 - `css()` method, 156–158
 - `dblclick()` function, 123–124
 - descendants, 162–164
 - element selector, 119–120
 - `empty()` method, 152–153
 - `eq()` method, 173–174
 - events, 122–125
 - `fading()` function, 128–129
 - filtering, 171–174
 - `find()` method, 163–164
 - `first()` method, 171–172
 - getting started, 243
 - `hide()` and `show()` methods, 126–127
 - home page, 243–247
 - `id` attribute, 120–121
 - `last()` method, 172–173
 - `mouseenter()` function, 124–125

mouseleave() function, 125
 nextAll() method, 166–167
 next() method, 165–166
 nextUntil() method, 167–168
 parent() method, 159–160
 parents() method, 160–161
 parentsUntil() method, 161–162
 prepend() method, 146–147
 prevAll() method, 169–170
 prev() method, 168–169
 prevUntil() method, 170–171
 removeclass(), 155
 remove() method, 149–151
 selectors, 119–122
 siblings, 164–171
 siblings() method, 164–165
 slideToggle() function, 131–132
 sliding() function, 130–131
 stop() method, 135–136
 syntax, 118–119
 toggleclass(), 156
 toggle() function, 128
 traversing, 158–162

L

Lambda expressions
 with loops, 473
 with multiple parameter, 469–472
 return keyword, 470–472
 with threads, 473–474
 without parameter, 468–469
 language syntax, 299
 Last In, First Out (LIFO) principle, 374
 LinkedHashMap, 382–383
 LinkedHashSet, 384–385
 LinkedList, 389–390
 login–registration–ordering flowchart, 25–26
 Loser Tree, 452

M

Model View Controller (MVC), 11–12
 multithreading, 488–493, 506–511
 MySQL, installing, 637–638

N

Node.js, 1, 11
 NoSQL databases, 15
 nullified reference variables, 420–421

O

object-oriented programming language (OOP)
 abstraction, 267–268

application, 350–351
 classes, 351
 cohesion, 357
 coupling, 357
 encapsulation, 268
 future of, 359–360
 HAS-A relationships, 267
 history of, 337–338
 implementation in Java, 357
 inheritance, 269
 interface, 351
 IS-A relationships, 266–267
 objects, 350–351
 objects as solutions, 357–358
 overloading, 354–356
 overriding, 353–354, 356
 polymorphism, 269–270
 principles, 338–349
 in real-world scenarios and comparative solutions, 360
 tips for implementing, 358–359
 object-oriented programming (OOP) languages, 15
 Object Relational Mapping (ORM) tool, 15–16
 operands, 313
 order processing flowchart, 27–28

P

PHP, 11
 POJOs, 611–618
 polymorphism, 345–349
 advantages of, 349
 dynamic, 348
 static, 348
popper.js, 251
 POST endpoints, 32–34
 printOutput() method, 348
 PriorityQueue, 391–392
 project planning, 19
 Python, 1, 11

R

reactive programming, 485–488
 Read–Eval–Print–Loop (REPL) functional, 287
 reassigning reference variables, 420
 relational databases, 15
 Ruby on Rails, 1

S

Simple Object Access Protocol (SOAP), 13–14
 splitting strings, 430–433
 Spring framework
 architecture, 525–527
 aspect-oriented programming (AOP), 523–524
 chain of Resolvers, 528

CRUD application, 543–549
 dependency injection, 518–523
 form tags, 539
 form text field, 539–543
 interception, 528
 inversion of control, 517
 model interface, 535–537
 multiple controllers, 533–535
 multiple view pages, 531–533
 MVC, 527–528
 @RequestParam annotation, 537–539
 uploading files, 549–551
 validation for regular expression, 555–558
 validation in, 551–555
 validation with numbers, 558–561
 view of resolution, 529–531
 Spring Starter Project, 602–610
 Spring Tool Suite 4, 600–601
 static polymorphism, 348
 Stream API, 392–393
 Structured Query Language (SQL), 15
 subtrees, 449
 syntax, 299

T

three-tier architecture, 2
 Tournament Tree, 452
 TreeMap, 381–382
 tree related data structures, 448–452
 TreeSet, 385–386

U

UML class diagram, 22, 24
 unboxing process, 311–313
 undirected graph, 456
 unreachable objects, 418–419

V

vendor order processing flowchart, 28–29

W

web application development, 3
 web design, 2
 weighted graph, 454
 “while” loop statement, 329–330
 Winner Tree, 452
 World Wide Web Consortium (W3C), 4
 World Wide Web (WWW), 1
 wrapper classes, 309–311
 Character Wrapper, 309
 Constructors, 309
 conversion utility methods, 310–311
 creation of objects, 309–310
 outline of, 309
 primitive values, 309
 string representation, 309–310
 valueOf(String val), 310
 valueOf(String val, int radix), 310
 “Write Once, Run Anywhere” (WORA) principle, 259

About the Book

Full stack development is the development of both front-end and back-end of an application. This type of web development takes care of everything, from the conception of an idea to the real finished product. Today, it is very complex and expensive to have a specific specialist to work on different subsystems of web development process. For this reason, companies are demanding full stack developers who are able to work across multiple stacks.

Full Stack Java Development is a great way to start one's journey to becoming a full stack developer. Any student will find this book very helpful to learn about the essential technologies and ecosystem of web application development. Web technologies like HTML, JQuery, Bootstrap, Webservices, etc. are well explained in the book. The exploration is not just limited to theoretical knowledge as the book also has add-ons to improve one's understanding of the subject. This book will greatly help students prepare to become full stack developers.

Salient Features

- Introduction of **Full Stack Development** and the technologies involved.
- Familiarization with front-end development, including **HTML**, **jQuery**, **CSS**, and **Bootstrap**.
- Exploration of **Java Programming Language**.
- Examination of **Spring** and **Hibernate** concepts.
- Study on back-end development, including **MVC**, **REST**, and **webservices APIs**.
- Numerous **Multiple-Choice Questions**, **Review Questions**, **Exercises**, and **Project Assignments** provided.
- **Recommended Readings** provide links for further study.

For better learning, comprehension, and retention, the book is supported by the following:

- **QUICK CHALLENGE** stimulates a student's level of understanding with an out-of-the-box question.
- **FACT ALERT** explains some of the important truths.
- **FLASH QUIZ** tests a student's knowledge on the concept discussed.

Codes and Starter Kit available at
<https://github.com/MayurRamgir/FullStackJavaByWiley>

Scan code to
Access **Videos**



follow us on



facebook.com/wileyindia



twitter.com/wileyindiapl



linkedin.com/in/wileyindia

READER LEVEL
Beginner to Advanced

SHELVING CATEGORY
Engineering

WILEY

Wiley India Pvt. Ltd.
Customer Care +91 120 6291100
csupport@wiley.com
www.wileyindia.com
www.wiley.com

ISBN: 978-81-265-1991-0

