INTERNSHIP REPORT AT
COGNIZANT TECHNOLOGY SOLUTIONS

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology

in

MECHANICAL ENGINEERING

By

MUFADDAL LOKHANDWALA (17BME0313)

June,2021

SCHOOL OF MECHANICAL ENGINEERING(SMEC), Vellore

Cognizant

12-Jan-2021
Mufaddal Lokhandwala
B.Tech/B.E. Mechanical Engineering
VIT University, Vellore

Dear Mufaddal,

Further to our offer for the position of Programmer Analyst Trainee and in response to your confirmation into the Internship opportunity we had extended, we are pleased to offer you an **Internship** with us for **a period of 3 to 6 months**, during which you will be offered a stipend Amount of **INR 12000/-** per month based on the Internship performance and completion.
Actual Internship dates and duration would be based on the business demand aligned skill tracks offered to you and would be shortly communicated to you.
Cognizant Internship being a pre joining skill and capability development program, it would form a critical part of your employment with Cognizant.
You will undergo a learning curriculum as per the learning track assigned to you. The learning path will include in-depth sessions, hands on exercise and project work. There will also be series of webinars, quizzes, SME interactions, mentor connects, code challenges, assessments etc. to accelerate your learning. The performance during Internship would be monitored through formal evaluations.
The Cognizant Internship completion would qualify as the entry criteria to your post joining training program and would be used as basis towards your allocation to projects/roles.
Prior to joining Cognizant, you must successfully complete the prescribed Internship program. In event of non-completion of the Internship, Cognizant may at its sole discretion revoke this offer of employment.
Please also note that:

- The Internship Training will be done from Monday through Friday for 8 hours from 9 am to 6 pm (IST).
- Interns are covered under Cognizant's calendar holidays of the respective location of internship and you would need to adhere with attendance requirements. Pre-approvals are to be sought towards unavoidable leave or break requests from the program.
- There would be zero tolerance to plagiarisms and misconduct during the internship.
- You would be required to ensure timely completion and submission of assignments, project work and preparation required prior to the sessions.
- You may be required, to travel to other locations within India if there is a business need as per your internship plan
- Cognizant reserves clauses regarding IT infra if applicable and access to information and material of Cognizant during the period and could modify or amend the Cognizant GenC program terms and conditions from time to time

At the time of your reporting for the internship, you will be required to sign a Non - Disclosure Agreement with the company. During the course of your Internship and after completion of the same, you are required to maintain strictest confidentiality with respect to company proprietary or products that you access or come into contact with, during your project as an Intern, at all times as per our Policy. Use of company proprietary information or products shall not be made without prior permission from the concerned authority.
You will also be required to submit the following documents at the time of reporting;

- Photocopy of your Passport & Visa

DECLARATION

I hereby declare that the thesis entitled "INTERNSHIP REPORT AT COGNIZANT TECHNOLOGY SOLUTIONS" submitted by me, for the award of the degree of Bachelor of Technology in Mechanical Engineering is a record of Bonafide work carried out by me under the supervision of my coach. I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Work From Home

Date: 19-06-2021

Signature Of The Candidate

ABSTRACT

This report summarizes all the things that I had done in my internship with the duration of 4 months which is considered as my 8th semester capstone project(PAT internship) for my course completion in Mechanical Engineering.

Here are some of the Technologies that I learned in the internship:

- HTML, CSS, JAVASCRIPT
- JAVA
- MYSQL
- Spring framework in JAVA
- Spring MVC
- Spring ORM
- Spring Boot
- React
- Combining front-end(React) with the back-end(Spring)

The topics mentioned above as a whole are considered as Full Stack Web Development. So by learning these technologies in CTS. In CTS we were assigned a cohort of around 20 people and in one particular cohort (INTADM21AJ002) we were assigned POD's (POD2) which contains 4-5 people of that cohort. In order to complete our Internship we have to do MFPE (My First Project Engagement) which is nothing but the project that we have to do using the technologies learned during the internship.

In this Report I gathered all the project details and activities that I did in my 4 month long internship and the behavioural lessons that I learned which helped me boost my career.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to my mentor, coach and Trainer from Cognizant India for their constant guidance, continual encouragement, and understanding; more than all, they taught me patience in my endeavour. My association with them is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Information Technology.

I would like to express my gratitude to Mr. G. Viswanathan, Mr. Shankar Viswanathan and Dr. Anand A Samuel for providing an environment to work in and for his inspiration during the tenure of the course.

In a jubilant mood, I express my heartfelt gratitude to Dr. Murugan M (Professor Grade 1), former HOD of Mechanical Engineering, and Dr. Jeevanantham A.K. (Higher academic Grade), current HOD of Mechanical Engineering under SMEC, VIT Vellore, and all of his teaching staff and members working as members of our university for their non-self-centred enthusiasm coupled with time enlightenment. I'd like to express my gratitude to my parents for their unwavering support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore

Date: 10-06-2021 Signature Of the Candidate

TABLE OF CONTENTS

CONTENTS ***Page No***

1. INTRODUCTION

1.1 About the Company: Cognizant (CTS)

Cognizant is an American multinational technology company that provides business consulting, information technology and outsourcing services.It is headquartered in Teaneck, New Jersey, United States. Cognizant is part of the NASDAQ-100 and trades under CTSH. It was founded as an in-house technology unit of Dun & Bradstreet in 1994, and started serving external clients in 1996.After a series of corporate reorganizations there was an initial public offering in 1998. Cognizant had a period of fast growth during the 2000s and became a *Fortune* 500 company in 2011; as of 2021 it's at position 185.

Cognizant began as Dun & Bradstreet Software (DBSS), established as Dun & Bradstreet's in-house technology unit focused on implementing large-scale IT projects for Dun & Bradstreet businesses. In 1996, the company started pursuing customers beyond Dun & Bradstreet. In 1996, Dun & Bradstreet spun off several of its subsidiaries including Erisco, IMS International, Nielsen Media Research, Pilot Software, Strategic Technologies and DBSS, to form a new company called Cognizant Corporation, headquartered in Chennai, India. Three months later, in 1997, DBSS renamed itself to Cognizant Technology Solutions. In July 1997, Dun & Bradstreet bought Satyam's 24% stake in DBSS for $3.4 million. Headquarters were moved to the United States, and in March 1998, Kumar Mahadeva was named CEO. Operating as a division of the Cognizant Corporation, the company focused on Y2K-related projects and web development.

In 1998, the parent company, Cognizant Corporation, split into two companies: IMS Health and Nielsen Media Research. After this restructuring, Cognizant Technology Solutions became a public subsidiary of IMS Health. In June 1998, IMS Health partially spun off the company, conducting an initial public offering of the Cognizant stock. The company raised $34 million, less than what the IMS Health underwriters had hoped. They earmarked the money for debt payments and upgrading company offices.

In 2021, 792,000 employees including contractors in Hyderabad Cognizant, which instituted a poison pill provision to prevent hostile takeover attempts. Kumar Mahadeva resigned as the CEO in 2003, and was replaced by Lakshmi Narayanan. Gradually, the company's services portfolio expanded across the IT services landscape and into business process outsourcing (BPO) and business consulting. Lakshmi Narayanan was succeeded by Francisco D'Souza in 2006. Cognizant experienced a period of fast growth during the 2000s, as reflected

by its appearance in *Fortune* magazine's "100 Fastest-Growing Companies" list for ten consecutive years from 2003 to 2012.

1.2 About the Department(ADM)

During my internship, I was part of the Application Development Management(ADM) team in which I have been taught all the technologies required to develop Full Stack Applications.

1.2.1 Ice Breaker Sessions

During this period all of the interns got to know about the work ethics and values of the company through a platform called cognizant-learn where we have been given some online courses which have to be completed in the initial week. We also had workshops on Agile and Devops which helped us in making our project using Agile methodology.

1.2.2 Core Program Fundamentals

HTML:

The **HyperText Markup Language**, or **HTML** is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by *tags*, written using angle brackets. Tags such as <**img** /> and <**input** /> directly introduce content into the page. Other tags such as <**p**> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

Fig1:HTML

CSS:

Cascading Style Sheets (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate **.css** file which reduces complexity and repetition in the structural content as well as enabling the **.css** file to be cached to improve the page load speed between the pages that share the file and its formatting.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

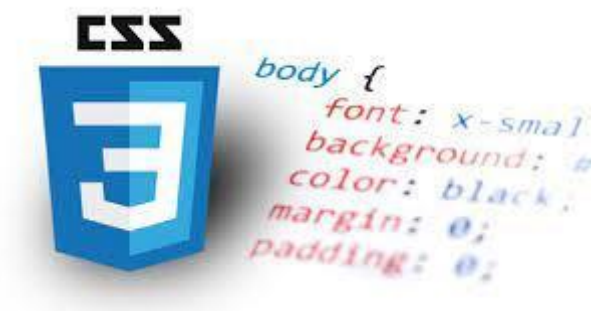


Fig2: CSS

JAVASCRIPT:

JavaScript often abbreviated as **JS**, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm .It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on the user's device. As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O. JavaScript engines were originally used only in web browsers, but they are now core components of other software systems, most notably servers and a variety of applications. Although there are similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design.

MySQL:

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the

relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses. MySQL was owned and sponsored by the Swedish company MySQL AB, which was bought by Sun Microsystems (now Oracle Corporation). In 2010, when Oracle acquired Sun, Widenius forked the open-source MySQL project to create MariaDB.

MySQL has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often, MySQL is used with other programs to implement applications that need relational database capability. MySQL is a component of the LAMP web application software stack (and others), which is an acronym for *Linux*, *Apache*, *MySQL*, *Perl/PHP/Python*. MySQL is used by many database-driven web applications, including Drupal, Joomla, phpBB, and WordPress.

JAVA:

Java is a high-level, class-based, object-oriented programming language that is designed have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers *write once, run anywhere* (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but has fewer low-level facilities than either of them. The Java runtime provides dynamic capabilities (such as reflection and runtime code modification) that are typically not available in traditional compiled languages. As of 2019,

Java was one of the most popular programming languages in use according to GitHub, particularly for client-server web applications, with a reported 9 million developers.

Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle) and released in 1995 as a core component of Sun Microsystems' Java platform. The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun had relicensed most of its Java technologies under the GNU General Public License. Oracle offers its own HotSpot Java Virtual Machine, however the official reference implementation is the OpenJDK JVM which is free open source software and used by most developers and is the default JVM for almost all Linux distributions.

As of March 2021, the latest version is Java 16, with Java 11, a currently supported long-term support (LTS) version, released on September 25, 2018. Oracle released the last zero-cost public update for the legacy version Java 8 LTS in January 2019 for commercial use, although it will otherwise still support Java 8 with public updates for personal use indefinitely. Other vendors have begun to offer zero-cost builds of OpenJDK 8 and 11 that are still receiving security and other upgrades.

1.2.3 Deep Learnings

SPRING:

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Features

- **Core technologies:** dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP.
- **Testing:** mock objects, TestContext framework, Spring MVC Test, WebTestClient.
- **Data Access:** transactions, DAO support, JDBC, ORM, Marshalling XML.
- **Spring MVC and Spring WebFlux web frameworks.**
- **Integration:** remoting, JMS, JCA, JMX, email, tasks, scheduling, cache.
- **Languages:** Kotlin, Groovy, dynamic languages.

Maven:

Maven, a Yiddish word meaning *accumulator of knowledge*, began as an attempt to simplify the build processes in the Jakarta Turbine project. There were several projects, each with their own Ant build files, that were all slightly different. JARs were checked into CVS. We wanted a standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information, and a way to share JARs across several projects.

The result is a tool that can now be used for building and managing any Java-based project. We hope that we have created something that will make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

Junit and Code Quality:

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit.

JUnit is linked as a JAR at compile-time. The latest version of the framework, JUnit 5, resides under package org.junit.jupiter. Previous versions JUnit 4 and JUnit 3 were under packages org.junit and junit.framework, respectively.

A research survey performed in 2013 across 10,000 Java projects hosted on GitHub found that JUnit (in a tie with slf4j-api), was the most commonly included external library. Each library was used by 30.7% of projects.

Spring MVC:

The Spring Framework features its own model–view–controller (MVC) web application framework, which wasn't originally planned. The Spring developers decided to write their own Web framework as a reaction to what they perceived as the poor design of the (then) popular Jakarta Struts Web framework, as well as deficiencies in other available frameworks. In particular, they felt there was insufficient separation between the presentation and request handling layers, and between the request handling layer and the model.

Like Struts, Spring MVC is a request-based framework. The framework defines strategy interfaces for all of the responsibilities that must be handled by a modern request-based framework. The goal of each interface is to be simple and clear so that it's easy for Spring MVC users to write their own implementations, if they so choose. MVC paves the way for cleaner front end code. All interfaces are tightly coupled to the Servlet API. This tight coupling to the Servlet API is seen by some as a failure on the part of the Spring developers to offer a high-level abstraction for Web-based applications. However, this coupling makes sure

that the features of the Servlet API remain available to developers while also offering a high abstraction framework to ease working with it.

The DispatcherServlet class is the front controller of the framework and is responsible for delegating control to the various interfaces during the execution phases of an HTTP request.

Spring Boot:

Spring Boot is Spring's convention-over-configuration solution for creating stand-alone, production-grade Spring-based Applications that you can "just run". It is preconfigured with the Spring team's "opinionated view" of the best configuration and use of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration. Features:

- Create stand-alone Spring applications
- Embed Tomcat or Jetty directly (no need to deploy WAR files)
- Provide opinionated 'starter' Project Object Models (POMs) to simplify your Maven configuration
- Automatically configure Spring whenever possible
- Provide production-ready features such as metrics, health checks and externalized configuration
- Absolutely no code generation and no requirement for XML configuration.

1.2.4 Niche Skills

RestFul Web Services:

Representational state transfer (**REST**) is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web. REST defines a set of constraints for how the architecture of an Internet-scale distributed hypermedia system, such as the Web, should behave. The REST architectural style emphasises the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems. REST has been employed throughout the software industry and is a widely accepted set of guidelines for creating stateless, reliable web services.

Any web service that obeys the REST constraints is informally described as **RESTful**. Such a web service must provide its Web resources in a textual representation and allow them to be read and modified with a stateless protocol and a predefined set of operations. This approach allows the greatest interoperability between clients and servers in a long-lived Internet-scale environment which crosses organisational (trust) boundaries.

Git:

Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Since 2005, Junio Hamano has been the core maintainer. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.

JQuery:

JQuery is a JavaScript library designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax. It is free, open-source software using the permissive MIT License. As of May 2019, jQuery is used by 73% of the 10 million most popular websites. Web analysis indicates that it is the most widely deployed JavaScript library by a large margin, having at least 3 to 4 times more usage than any other JavaScript library.

jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plug-ins on top of the JavaScript library. This enables developers to create abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets. The modular approach to the jQuery library allows the creation of powerful dynamic web pages and Web applications.

The set of jQuery core features—DOM element selections, traversal and manipulation—enabled by its *selector engine* (named "Sizzle" from v1.3), created a new "programming style", fusing algorithms and DOM data structures. This style influenced the architecture of other JavaScript frameworks like YUI v3 and Dojo, later stimulating the creation of the standard *Selectors API*. Later, this style has been enhanced with a deeper algorithm-data fusion in an heir of jQuery, the D3.js framework.

BootStrap:

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components. As of April 2021, Bootstrap is the tenth most starred project on GitHub, with more than 150,000 stars.

Microservices:

Microservice architecture – a variant of the service-oriented architecture (SOA) structural style – arranges an application as a collection of loosely-coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight.

There is no single definition for microservices. A consensus view has evolved over time in the industry. Some of the defining characteristics that are frequently cited include:

- Services in a microservice architecture (MSA) are often processes that communicate over a network to fulfil a goal using technology-agnostic protocols such as HTTP.
- Services are organized around business capabilities.
- Services can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best.[5]
- Services are small in size, messaging-enabled, bounded by contexts, autonomously developed, independently deployable,[6][5] decentralized and built and released with automated processes.[6]

AWS:

Amazon Web Services (**AWS**) is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. These cloud computing web services provide a variety of basic abstract technical infrastructure and distributed computing building blocks and tools. One of these services is Amazon Elastic Compute Cloud (EC2), which allows users to have at their disposal a virtual cluster of computers, available all the time, through the Internet. AWS's version of virtual computers emulates most of the attributes of a real computer, including hardware central processing units (CPUs) and graphics processing units (GPUs) for processing; local/RAM memory; hard-disk/SSD storage; a choice of operating systems; networking; and pre-loaded application software such as web servers, databases, and customer relationship management (CRM).

The AWS technology is implemented at server farms throughout the world, and maintained by the Amazon subsidiary. Fees are based on a combination of usage (known as a "Pay-as-you-go" model), hardware, operating system, software, or networking features chosen by the subscriber required availability, redundancy, security, and service options. Subscribers can pay for a single virtual AWS computer, a dedicated physical computer, or clusters of either. As part of the subscription agreement,[10] Amazon provides security for subscribers' systems. AWS operates from many global geographical regions including 6 in North America.[11]

React:

React (also known as **React.js** or **ReactJS**) is an open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

1.3 Evaluation

1.3.1 Continuous Learning

Continuous learning includes Hands-On exercises that would be a part of regular learning.

 Weightage: ~5% on your Performance health score.

 Mandate: 100% Hands-On completion.

1.3.2 Assess Type 1

Assess type 1 predominantly includes Coding Challenge

 Weightage: ~10% on your Performance health score.

 Prerequisite: 100% Hands-on completion.

 Mandate: Pass mark is 70% for completion.

1.3.3 Assess Type 2

Assess type 2 includes Integrated Capability Test, Knowledge Based Assessment (KBA) and Skill Based Assessment (SBA).

 Weightage: ~30% on your Performance health score.

 Prerequisite: 100% Hands-on completion and attempted all type 1 and type 2 assessments until this point.

 Mandate: Pass mark is 70% for completion.

1.3.4 Business Aligned Project

In the Business Aligned Project, end to end technical capability and agile concepts will be covered.

 Weightage: ~40% on your Performance health score. Mandate: Pass mark is 70% for completion.

2. TECHNICAL TRAINING – Stage 1

Stage 1 contains total of 5 weeks

2.1 Week 1 Of Training:

2.1.1 Html:

In its simplest form, following is an example of an HTML document

```
<!DOCTYPE html>
<html>

   <head>
      <title>This is document title</title>
   </head>

   <body>
      <h1>This is a heading</h1>
      <p>Document content goes here.....</p>
   </body>

</html>
```

Html Tags:

Sr.No	Tag & Description
1	**<!DOCTYPE...>** This tag defines the document type and HTML version.
2	**<html>** This tag encloses the complete HTML document and mainly comprises document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.

3	**<head>** This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.
4	**<title>** The <title> tag is used inside the <head> tag to mention the document title.
5	**<body>** This tag represents the document's body which keeps other HTML tags like <h1>, , <p> etc.
6	**<h1>** This tag represents the heading.
7	**<p>** This tag represents a paragraph.

Attributes:

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts – a **name** and a **value**

- The **name** is the property you want to set. For example, the paragraph **<p>** element in the example carries an attribute whose name is **align**, which you can use to indicate the alignment of paragraph on the page.
- The **value** is what you want the value of the property to be set and always put within quotations. The below example shows three possible values of align attribute: **left, center** and **right**.

Attribute names and attribute values are case-insensitive. However, the World Wide Web Consortium (W3C) recommends lowercase attributes/attribute values in their HTML 4 recommendation.

```
<!DOCTYPE html>
<html>

   <head>
      <title>Align Attribute  Example</title>
   </head>

   <body>
      <p align = "left">This is left aligned</p>
      <p align = "center">This is center aligned</p>
      <p align = "right">This is right aligned</p>
   </body>

</html>
```

Core Attributes

The four core attributes that can be used on the majority of HTML elements (although not all) are –

- Id
- Title
- Class
- Style

2.1.2 CSS

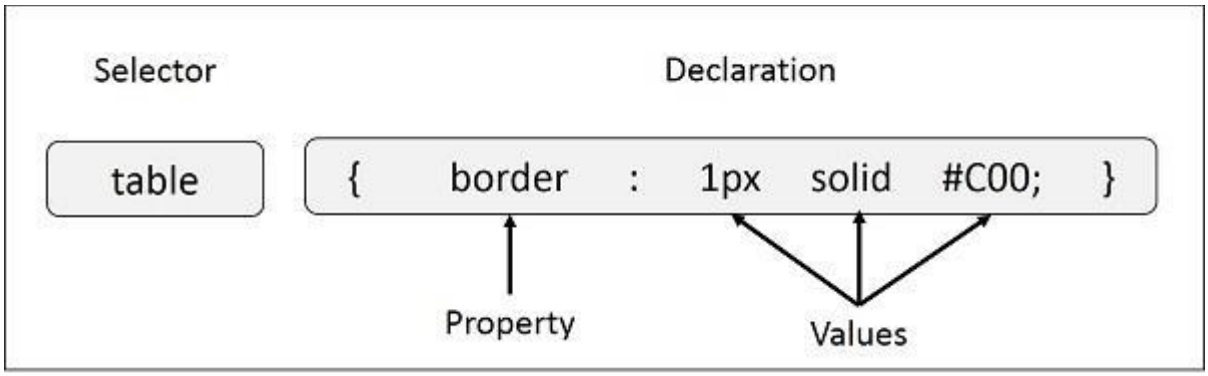
A CSS comprises style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts –

- **Selector** – A selector is an HTML tag at which a style will be applied. This could be any tag like <hl> or <table> etc.
- **Property** – A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be *color*, *border* etc.

- **Value** – Values are assigned to properties. For example, *color* property can have value either *red* or *#F1F1F1* etc.

You can put CSS Style Rule Syntax as follows –

selector { property: value }



Example – You can define a table border as follows –

table{ border :1px solid #C00; }

Here table is a selector and border is a property and given value *1px solid #C00* is the value of that property. You can define selectors in various simple ways based on your comfort. Let me put these selectors one by one.

Multiple Style Rules

You may need to define multiple style rules for a single element. You can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example –

```
h1 {
  color: #36C;
  font-weight: normal;
  letter-spacing: .4em;
  margin-bottom: 1em;
  text-transform: lowercase;
}
```

Here all the property and value pairs are separated by a **semicolon (;)**. You can keep them in a single line or multiple lines. For better readability, we keep them in separate lines.

For a while, don't bother about the properties mentioned in the above block. These properties will be explained in the coming chapters and you can find complete detail about properties in CSS References

2.1.3 JAVASCRIPT

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
  JavaScript code
</script>
```

The script tag takes two important attributes –

- **Language** – This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like –

```
<script language = "javascript" type = "text/javascript">
  JavaScript code
</script>
```

Your First JavaScript Code

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment

as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>
  <body>
    <script language = "javascript" type = "text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>
  </body>
</html>
```

This code will produce the following result –

Hello World!

2.2 Week 2 Of Training:

2.2.1 MySQL

MySQL Queries

A list of commonly used MySQL queries to create database, use database, create table, insert record, update record, delete record, select record, truncate table and drop table are given below.

1) MySQL Create Database

MySQL create database is used to create databases. For example

create database db1;

2) MySQL Select/Use Database

MySQL database is used to select databases. For example use db1;

3) MySQL Create Query

MySQL create query is used to create a table, view, procedure and function. For example:

```
1. CREATE TABLE customers
2. (id int(10),
3.  name varchar(50),
```

4. city **varchar**(50),
5. **PRIMARY KEY** (id)

4) MySQL Alter Query

MySQL alter query is used to add, modify, delete or drop colums of a table. Let's see a query to add column in customers table:

1. **ALTER TABLE** customers
2. **ADD** age **varchar**(50);

5) MySQL Insert Query

MySQL insert query is used to insert records into tables. For example:

insert into customers **values**(101,'rahul','delhi');

6) MySQL Update Query

MySQL update query is used to update records of a table. For example:

update customers **set name**='bob', city='london' **where** id=101;

7) MySQL Delete Query

MySQL update query is used to delete records of a table from a database. For example:

delete from customers **where** id=101;

8) MySQL Select Query

Oracle select query is used to fetch records from the database. For example:

SELECT * **from** customers;

9) MySQL Truncate Table Query

MySQL update query is used to truncate or remove records of a table. It doesn't remove structure. For example:

truncate table customers;

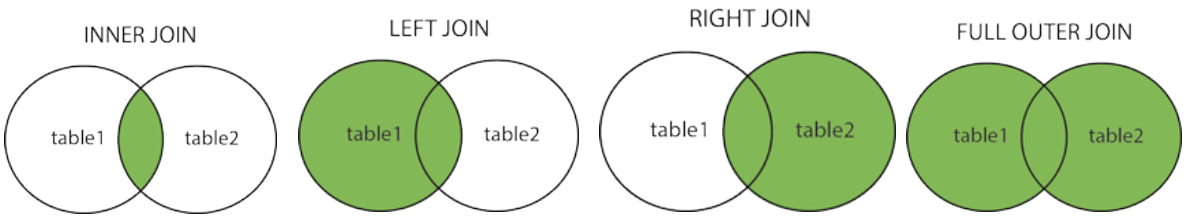
10) MySQL Drop Query

MySQL drop query is used to drop a table, view or database. It removes structure and data of a table if you drop it. For example: **drop table** customers;

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



2.3 Week 3,4,5 Of Training:

2.3.1 JAVA

Java is –

- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machines, rather into platform independent bytecode. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compilers in Java are written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.

- **Distributed** – Java is designed for the distributed environment of the internet.
- **Dynamic** – Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what class, object, methods, and instance variables mean.

- **Object** – Objects have states and behaviors. Example: A dog has states – color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.
- **Class** – A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
- **Methods** – A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** – Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

First Java Program

Let us look at a simple code that will print the words ***Hello World***.

Example

```
public class MyFirstJavaProgram {

  /* This is my first java program.
   * This will print 'Hello World' as the output
   */

  public static void main(String []args) {
```

```
    System.out.println("Hello World"); // prints Hello World
  }
}
```

Let's look at how to save the file, compile, and run the program. Please follow the subsequent steps –

- Open the notepad and add the code as above.
- Save the file as: MyFirstJavaProgram.java.
- Open a command prompt window and go to the directory where you saved the class. Assume it's C:\.
- Type 'javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption : The path variable is set).
- Now, type ' java MyFirstJavaProgram ' to run your program.
- You will be able to see ' Hello World ' printed on the window.

Output

```
C:\> javac MyFirstJavaProgram.java
C:\> java My First Java Program
Hello World
```

Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** – Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** – For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner world's first letter should be in Upper Case.

  Example: *class MyFirstJavaClass*
- **Method Names** – All method names should start with a Lowercase letter. If several words are used to form the name of the method, then each inner world's first letter should be in Upper Case.

Example: *public void myMethodName()*

- **Program File Name** – Name of the program file should exactly match the class name.

  When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

  But please make a note that in case you do not have a public class present in the file then file name can be different than class name. It is also not mandatory to have a public class in the file.

  Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

- **public static void main(String args[])** – Java program processing starts from the main() method which is a mandatory part of every Java program.

Java Variables

Following are the types of variables in Java –

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static Variables)

Java Arrays

Arrays are objects that store multiple variables of the same type. However, an array itself is an object on the heap. We will look into how to declare, construct, and initialize in the upcoming chapters.

Java Keywords

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

abstract	assert	boolean	break

byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

Comments in Java

Java supports single-line and multi-line comments very similar to C and C++. All characters available inside any comment are ignored by the Java compiler.

Example

```
public class MyFirstJavaProgram {

   /* This is my first java program.
    * This will print 'Hello World' as the output
    * This is an example of multi-line comments.
    */

   public static void main(String []args) {
      // This is an example of single line comment
      /* This is also an example of a single line comment. */
      System.out.println("Hello World");
   }
}
```

Output

```
Hello World
```

Inheritance

In Java, classes can be derived from classes. Basically, if you need to create a new class and there is already a class that has some of the code you require, then it is possible to derive your new class from the already existing code.

This concept allows you to reuse the fields and methods of the existing class without having to rewrite the code in a new class. In this scenario, the existing class is called the **superclass** and the derived class is called the **subclass**.

Objects in Java

If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

If you compare the software object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Classes in Java

A class is a blueprint from which individual objects are created.

Following is a sample of a class.

Example

```
public class Dog {
  String breed;
  int age;
  String color;

  void barking() {
  }

  void hungry() {
  }

  void sleeping() {
  }
}
```

A class can contain any of the following variable types.

- **Local variables** – Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables** – Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be

accessed from inside any method, constructor or blocks of that particular class.

- **Class variables** – Class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Following are some of the important topics that need to be discussed when looking into classes of the Java Language.

Constructors

When discussing classes, one of the most important subtopics would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class. Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Following is an example of a constructor –

Example

```
public class Puppy {
  public Puppy() {
  }

  public Puppy(String name) {
    // This constructor has one parameter, name.
  }
}
```

Java also supports Singleton Classes where you would be able to create only one instance of a class.

Note – We have two different types of constructors. We are going to discuss constructors in detail in the subsequent chapters.

Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class –

- **Declaration** – A variable declaration with a variable name with an object type.
- **Instantiation** – The 'new' keyword is used to create the object.
- **Initialization** – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Following is an example of creating an object –

Example

```
public class Puppy {
  public Puppy(String name) {
    // This constructor has one parameter, name.
    System.out.println("Passed Name is :" + name );
  }

  public static void main(String []args) {
    // Following statement would create an object myPuppy
    Puppy myPuppy = new Puppy( "tommy" );
  }
}
```

If we compile and run the above program, then it will produce the following result –

Output

```
Passed Name is :tommy
```

Accessing Instance Variables and Methods

Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path –

```
/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

Example

This example explains how to access instance variables and methods of a class.

```
public class Puppy {
  int puppyAge;

  public Puppy(String name) {
    // This constructor has one parameter, name.
    System.out.println("Name chosen is :" + name );
  }

  public void setAge( int age ) {
    puppyAge = age;
  }

  public int getAge( ) {
    System.out.println("Puppy's age is :" + puppyAge );
    return puppyAge;
  }

  public static void main(String []args) {
    /* Object creation */
    Puppy myPuppy = new Puppy( "tommy" );

    /* Call class method to set puppy's age */
    myPuppy.setAge( 2 );
```

```
    /* Call another class method to get puppy's age */
    myPuppy.getAge( );

    /* You can access instance variable as follows as well */
    System.out.println("Variable Value :" + myPuppy.puppyAge );
  }
}
```

If we compile and run the above program, then it will produce the following result –

Output

```
Name chosen is :tommy
Puppy's age is :2
Variable Value :2
```

Java Package

In simple words, it is a way of categorizing the classes and interfaces. When developing applications in Java, hundreds of classes and interfaces will be written, therefore categorizing these classes is a must as well as makes life much easier.

Import Statements

In Java if a fully qualified name, which includes the package and the class name is given, then the compiler can easily locate the source code or classes. Import statement is a way of giving the proper location for the compiler to find that particular class.

For example, the following line would ask the compiler to load all the classes available in directory java_installation/java/io –

```
import java.io.*;
```

Inheritance in Java:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java?

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

```
1. class Subclass-name extends Superclass-name
2. {
3.   //methods and fields
4. }
```

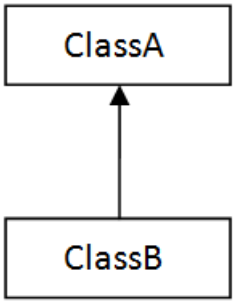
The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

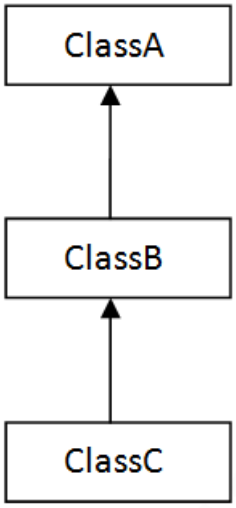
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

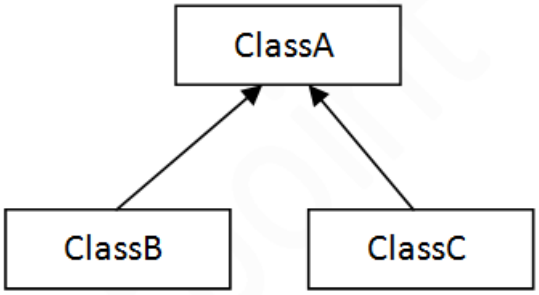
In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



1) Single

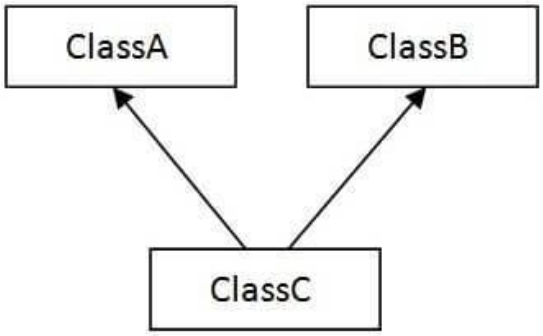


2) Multilevel

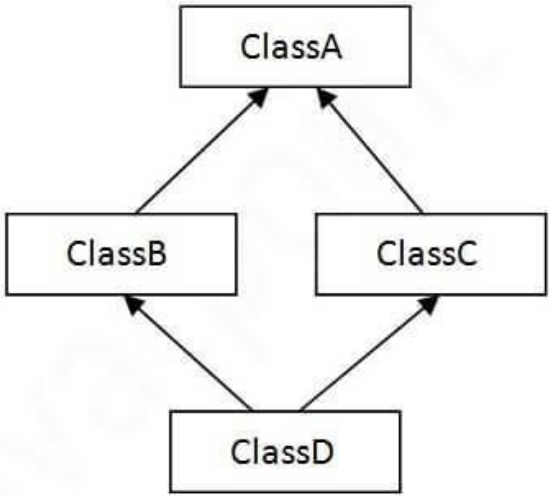


3) Hierarchical

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1) Method Overloading: changing no. of arguments

In this example, we have created two methods. The first add() method performs addition of two numbers and second add method performs addition of three numbers. In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
1. class Adder{
2. static int add(int a,int b){return a+b;}
3. static int add(int a,int b,int c){return a+b+c;}
4. }
5. class TestOverloading1{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(11,11,11));
9. }}
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and the second add method receives two double arguments.

```
1. class Adder{
2. static int add(int a, int b){return a+b;}
3. static double add(double a, double b){return a+b;}
4. }
5. class TestOverloading2{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(12.3,12.6));
9. }}
```

Method Overriding in Java

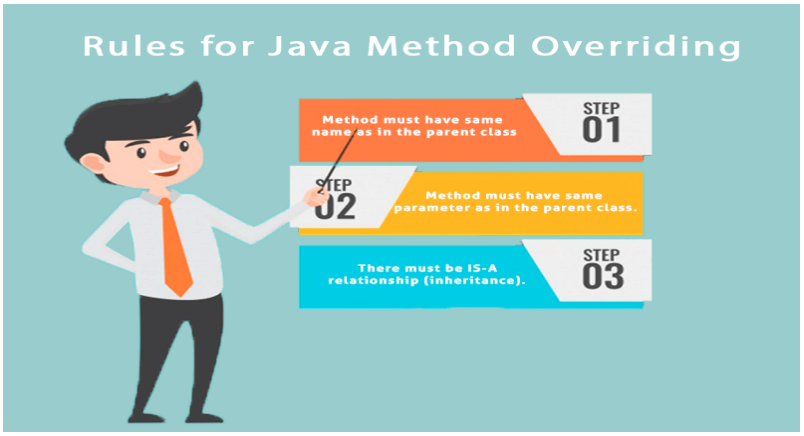
If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent classes, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).



Understanding the problem without method overriding

Let's understand the problem that we may face in the program if we don't use method overriding.

```
1. //Java Program to demonstrate why we need method overriding
2. //Here, we are calling the method of parent class with child
3. //class object.
4. //Creating a parent class
5. class Vehicle{
6.   void run(){System.out.println("Vehicle is running");}
7. }
8. //Creating a child class
9. class Bike extends Vehicle{
10.  public static void main(String args[]){
11.  //creating an instance of child class
12.  Bike obj = new Bike();
13.  //calling the method with child class instance
14.  obj.run();
15.  }
16. }
```

Problem is that I have to provide a specific implementation of the run() method in subclass that is why we use method overriding. Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

```
1. //Java Program to illustrate the use of Java Method Overriding
2. //Creating a parent class.
3. class Vehicle{
4.   //defining a method
5.   void run(){System.out.println("Vehicle is running");}
6. }
7. //Creating a child class
8. class Bike2 extends Vehicle{
9.   //defining the same method as in the parent class
10.  void run(){System.out.println("Bike is running safely");}
11.  public static void main(String args[]){
12.  Bike2 obj = new Bike2();//creating object
13.  obj.run();//calling method
14.  }
15. }
```

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it. Ways to achieve Abstraction

There are two ways to achieve abstraction in java

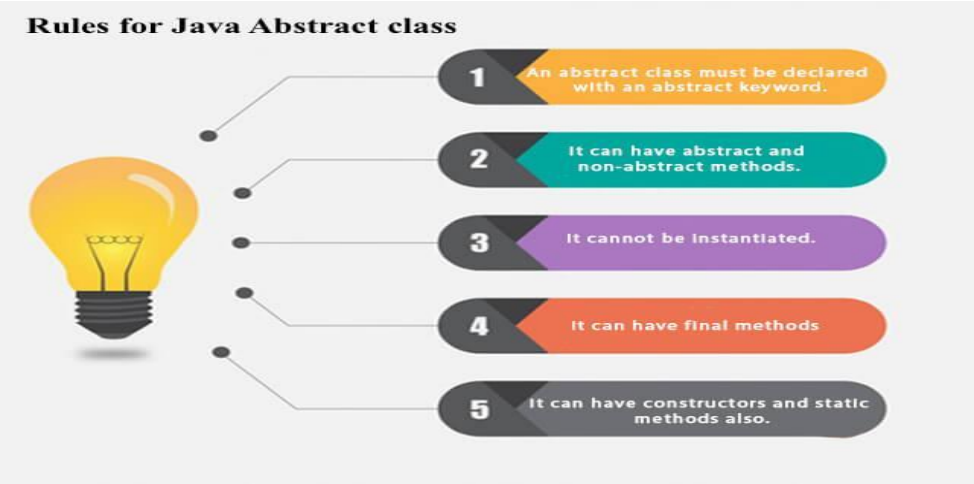
1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.



Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method bodies. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**. It cannot be instantiated just like the abstract class. Since Java 8, we can have **default and static methods** in an interface. Since Java 9, we can have **private methods** in an interface.

Why use the Java interface?

There are mainly three reasons to use interfaces. They are given below.

- It is used to achieve abstraction.

- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

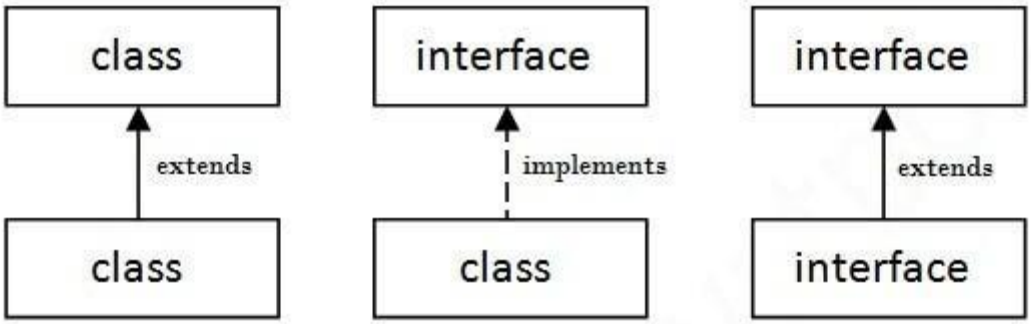
An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
1. interface <interface_name>{
2.
3.     // declare constant fields
4.     // declare methods that abstract
5.     // by default.
6. }
```

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
1. interface printable{
2. void print();
3. }
4. class A6 implements printable{
5. public void print(){System.out.println("Hello");}
6.
7. public static void main(String args[]){
8. A6 obj = new A6();
9. obj.print();
10. }
11. }
```

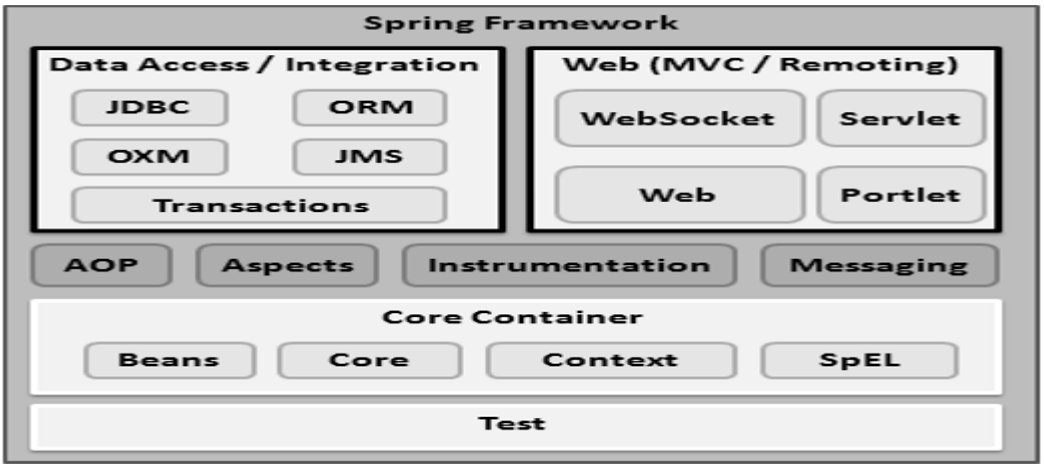
3.TECHNICAL TRAINING – STAGE 2

3.1 Week 6,7 Of Training:

3.1.1 Spring using Maven

Spring could potentially be a one-stop shop for all your enterprise applications. However, Spring is modular, allowing you to pick and choose which modules are applicable to you, without having to bring in the rest. The following section provides details about all the modules available in Spring Framework.

The Spring Framework provides about 20 modules which can be used based on an application requirement.



Core Container

The Core Container consists of the Core, Beans, Context, and Expression Language modules the details of which are as follows –

- The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
- The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
- The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows –

- The **JDBC** module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
- The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
- The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service **JMS** module contains features for producing and consuming messages.
- The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules the details of which are as follows –

- The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
- The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Miscellaneous

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules the details of which are as follows –

- The **AOP** module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.

- The **Aspects** module provides integration with AspectJ, which is again a powerful and mature AOP framework.
- The **Instrumentation** module provides class instrumentation support and class loader implementations to be used in certain application servers.
- The **Messaging** module provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
- The **Test** module supports the testing of Spring components with JUnit or TestNG frameworks.

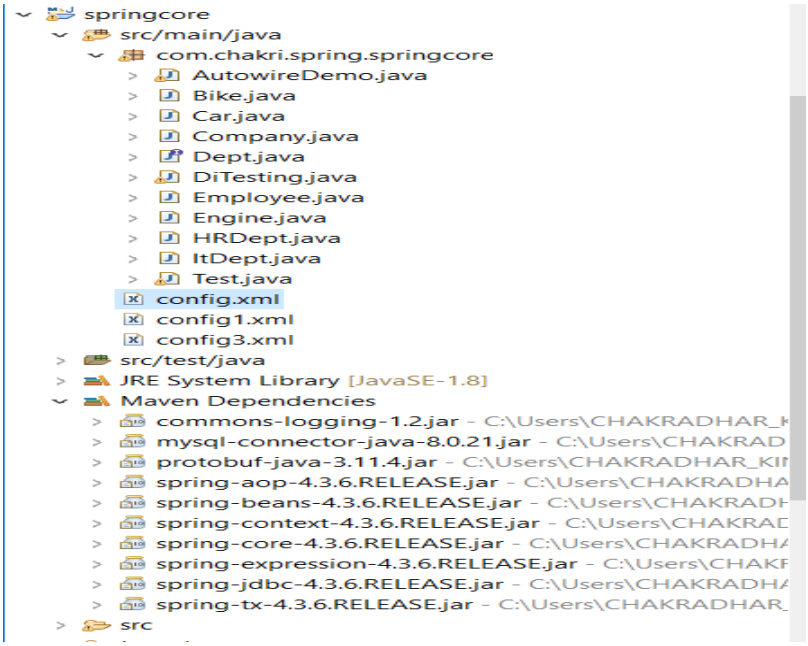
To Run a Spring Application

Now let us proceed to write a simple Spring Application, which will print "Hello World!" or any other message based on the configuration done in the Spring Beans Configuration file.

Step 1 - Create Java Project

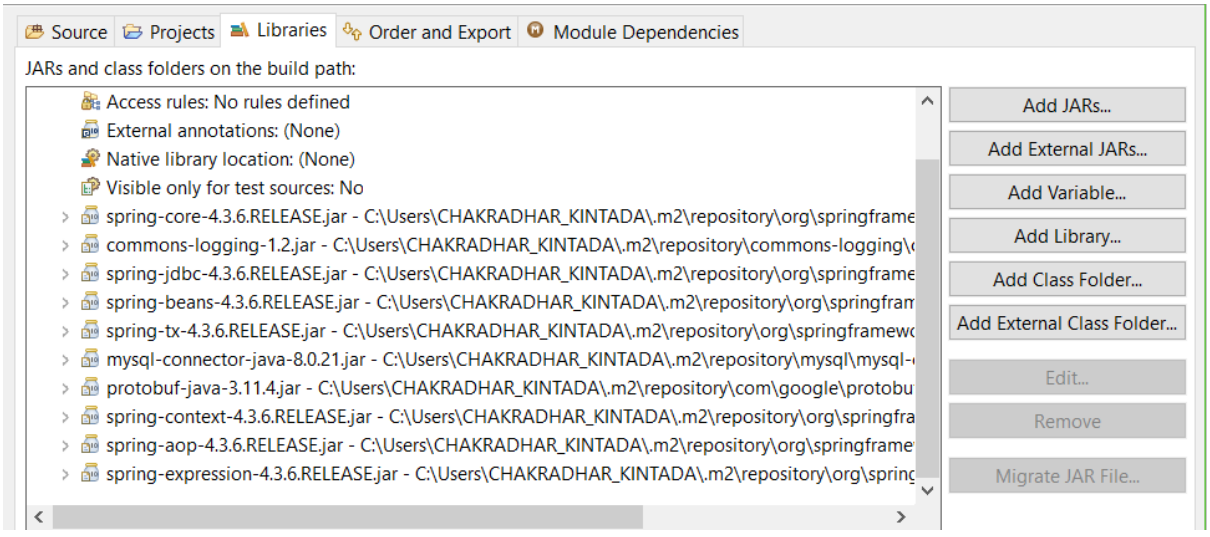
The first step is to create a simple Java Project using Eclipse IDE. Follow the option **File → New → Project** and finally select **Java Project** wizard from the wizard list. Now name your project as **springcore** using the wizard window as follows –

Once your project is created successfully, you will have the following content in your **Project Explorer** –



Step 2 - Add Required Libraries

As a second step let us add Spring Framework and common logging API libraries in our project. To do this, right-click on your project name **Springcore** and then follow the following option available in the context menu – **Build Path → Configure Build Path** to display the Java Build Path window as follows –



Now use **Add External JARs** button available under the **Libraries** tab to add the following core JARs from Spring Framework and Common Logging installation directories –(versions can be anything that picked from maven repository)

- commons-logging-1.1.1

- spring-aop-4.1.6.RELEASE
- spring-aspects-4.1.6.RELEASE
- spring-beans-4.1.6.RELEASE
- spring-context-4.1.6.RELEASE
- spring-context-support-4.1.6.RELEASE
- spring-core-4.1.6.RELEASE
- spring-expression-4.1.6.RELEASE
- spring-instrument-4.1.6.RELEASE
- spring-instrument-tomcat-4.1.6.RELEASE
- spring-jdbc-4.1.6.RELEASE
- spring-jms-4.1.6.RELEASE
- spring-messaging-4.1.6.RELEASE
- spring-orm-4.1.6.RELEASE
- spring-oxm-4.1.6.RELEASE
- spring-test-4.1.6.RELEASE
- spring-tx-4.1.6.RELEASE
- spring-web-4.1.6.RELEASE
- spring-webmvc-4.1.6.RELEASE
- spring-webmvc-portlet-4.1.6.RELEASE
- spring-websocket-4.1.6.RELEASE

Step 3 - Create Source Files

Now let us create actual source files under the **HelloSpring** project. First we need to create a package called **com.mufaddal.spring.springcore**. To do this, right click on **src** in package explorer section and follow the option – New → Package.

Next we will create **Employee.java** and **Test.java** files.

Here is the content of **Employee.java** `file –`

```
package com.anshu;.spring.springcore;

public class Employee {
private int id;
private String name;
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;
}
}
```

Following is the content of the second file **Test.java** –

```
package com.mufaddal.spring.springcore;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Test {

public static void main(String[] args) {

 ClassPathXmlApplicationContextcmt=new ClassPathXmlApplicationContext("config.xml");

                                    Employee emp = (Employee) cmt.getBean("emp");
```

```
                                        System.out.println("ID:"+emp.getId());

                                        System.out.println("Name:"+emp.getName());
                                                }
}
```

Following two important points are to be noted about the main program –

- The first step is to create an application context where we used framework API **ClassPathXmlApplicationContext()**. This API loads beans configuration file and eventually based on the provided API, it takes care of creating and initializing all the objects, i.e. beans mentioned in the configuration file.
- The second step is used to get the required bean using **getBean()** method of the created context. This method uses bean ID to return a generic object, which finally can be casted to the actual object. Once you have an object, you can use this object to call any class method.

Step 4 - Create Bean Configuration File

You need to create a Bean Configuration file which is an XML file and acts as a cement that glues the beans, i.e. the classes together. This file needs to be created under the **src** directory as shown in the following screenshot –

Usually developers name this file as **config.xml**, but you are independent to choose any name you like. You have to make sure that this file is available in CLASSPATH and use the same name in the main application while creating an application context as shown in Test.java file.

The Beans.xml is used to assign unique IDs to different beans and to control the creation of objects with different values without impacting any of the Spring source files. For example, using the following file you can pass any value for "message" variable and you can print different values of message without impacting HelloWorld.java and MainApp.java files. Let us see how it works –

```
<?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:p="http://www.springframework.org/schema/p"
                                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">
                                    <bean name="emp" class="com.spring.springcore.Employee">

                                                                    <property name="id">

                                                                    <value>20</value>

                                                                        </property>

                                                                <property name="name">

                                                                <value>prudhvi</value>

                                                                    </property>
                                                                    </bean>

</beans>
```

When Spring application gets loaded into the memory, Framework makes use of the above configuration file to create all the beans defined and assigns them a unique ID as defined in **<bean>** tag. You can use **<property>** tag to pass the values of different variables used at the time of object creation.

Step 5 - Running the Program

Once you are done with creating the source and beans configuration files, you are ready for this step, which is compiling and running your program. To do this, keep Test.Java file tab active and use either **Run** option available in the Eclipse IDE or use **Ctrl + F11** to compile and run your **Main** application. If everything is fine with your application, this will print the following message in Eclipse IDE's console −

Output:
ID : 20
Name: prudhvi

Congratulations, you have successfully created your first Spring Application. You can see the flexibility of the above Spring application by changing the value of "ID" property, ”Name” property and keeping both the source files unchanged.

When defining a <bean> you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be **prototype**. Similarly, if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be **singleton**.

The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.

Sr.No.	Scope & Description
1	**singleton** This scopes the bean definition to a single instance per Spring IoC container (default).
2	**prototype** This scopes a single bean definition to have any number of object instances.
3	**request** This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
4	**session** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

5	**global-session** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

The singleton scope

If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.

The default scope is always singleton. However, when you need one and only one instance of a bean, you can set the **scope** property to **singleton** in the bean configuration file, as shown in the following code snippet –

```
<!-- A bean definition with singleton scope -->
<bean id = "..." class = "..." scope = "singleton">
   <!-- collaborators and configuration for this bean go here -->
</bean>
```

The prototype scope

If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made. As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.

To define a prototype scope, you can set the **scope** property to **prototype** in the bean configuration file, as shown in the following code snippet –

```
<!-- A bean definition with prototype scope -->
<bean id = "..." class = "..." scope = "prototype">
   <!-- collaborators and configuration for this bean go here -->
</bean>
```

The Spring container can **autowire** relationships between collaborating beans without using <constructor-arg> and <property> elements, which helps cut down on the amount of XML configuration you write for a big Spring-based application.

Autowiring Modes

Following are the autowiring modes, which can be used to instruct the Spring container to use autowiring for dependency injection. You use the autowire attribute of the <bean/> element to specify **autowire** mode for a bean definition.

Sr.No	Mode & Description
1	**no** This is the default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing special to do for this wiring. This is what you already have seen in the Dependency Injection chapter.
2	**byName** Autowiring by property name. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byName* in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
3	**byType** Autowiring by property datatype. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byType* in the XML configuration file. It then tries to match and wire a property if its **type** matches with exactly one of the beans name in configuration file. If more than one such beans exists, a fatal exception is thrown.
4	**constructor** Similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.

5	**autodetect** Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*.

You can use **byType** or **constructor** autowiring mode to wire arrays and other typed-collections.

Limitations with autowiring

Autowiring works best when it is used consistently across a project. If autowiring is not used in general, it might be confusing for developers to use it to wire only one or two bean definitions. Though, autowiring can significantly reduce the need to specify properties or constructor arguments but you should consider the limitations and disadvantages of autowiring before using them.

Sr.No.	Limitations & Description
1	**Overriding possibility** You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.
2	**Primitive data types** You cannot autowire so-called simple properties such as primitives, Strings, and Classes.
3	**Confusing nature** Autowiring is less exact than explicit wiring, so if possible prefer using explict wiring.

Annotation Based Configuration

Starting from Spring 2.5 it became possible to configure the dependency injection using **annotations**. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider the following configuration file in case you want to use any annotation in your Spring application.

```
<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"
  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:annotation-config/>
   <bean id="hr" class="com.spring.springcore.HRDept"></bean>
   <bean id="it" class="com.spring.springcore.ItDept"></bean>
   <bean id="company" class="com.spring.springcore.Company" >
   </bean>
          </beans>
```

Once <context:annotation-config/> is configured, you can start annotating your code to indicate that Spring should automatically wire values into properties, methods, and constructors. Let us look at a few important annotations to understand how they work −

Sr.No.	Annotation & Description

1	@Required The @Required annotation applies to bean property setter methods.
2	@Autowired The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.
3	@Qualifier The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifying which exact bean will be wired.
4	JSR-250 Annotations Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.

Spring JDBC

While working with the database using plain old JDBC, it becomes cumbersome to write unnecessary code to handle exceptions, opening and closing database connections, etc. However, Spring JDBC Framework takes care of all the low-level details starting from opening the connection, preparing and executing the SQL statement, process exceptions, handle transactions and finally close the connection.

So what you have to do is just define the connection parameters and specify the SQL statement to be executed and do the required work for each iteration while fetching data from the database.

Spring JDBC provides several approaches and correspondingly different classes to interface with the database. I'm going to take classic and the most popular approach which makes use of **JdbcTemplate** class of the framework. This is the central framework class that manages all the database communication and exception handling.

JdbcTemplate Class

The JDBC Template class executes SQL queries, updates statements, stores procedure calls, performs iteration over ResultSets, and extracts returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package.

Instances of the *JdbcTemplate* class are *thread safe* once configured. So you can configure a single instance of a *JdbcTemplate* and then safely inject this shared reference into multiple DAOs.

A common practice when using the JDBC Template class is to configure a *DataSource* in your Spring configuration file, and then dependency-inject that shared DataSource bean into your DAO classes, and the JdbcTemplate is created in the setter for the DataSource.

Configuring Data Source

Let us create a database table **Student** in our database **TEST**. We assume you are working with MySQL database, if you work with any other database then you can change your DDL and SQL queries accordingly.

```
CREATE TABLE Student(
   ID   INT NOT NULL AUTO_INCREMENT,
   NAME VARCHAR(20) NOT NULL,
   AGE  INT NOT NULL,
   PRIMARY KEY (ID)
);
```

Now we need to supply a DataSource to the JDBC Template so it can configure itself to get database access. You can configure the DataSource in the XML file with a piece of code as shown in the following code snippet –

```
<bean id = "dataSource"
   class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name = "driverClassName" value = "com.mysql.jdbc.Driver"/>
   <property name = "url" value = "jdbc:mysql://localhost:3306/TEST"/>
   <property name = "username" value = "root"/>
   <property name = "password" value = "password"/>
</bean>
```

Data Access Object (DAO)

DAO stands for Data Access Object, which is commonly used for database interaction. DAOs exist to provide a means to read and write data to the database and they should expose this functionality through an interface by which the rest of the application will access them.

The DAO support in Spring makes it easy to work with data access technologies like JDBC, Hibernate, JPA, or JDO in a consistent way.

Executing SQL statements

Let us see how we can perform CRUD (Create, Read, Update and Delete) operation on database tables using SQL and JDBC Template objects.

Querying for an integer

```
String SQL = "select count(*) from Student";
int rowCount = jdbcTemplateObject.queryForInt( SQL );
```

Querying for a long

```
String SQL = "select count(*) from Student";
long rowCount = jdbcTemplateObject.queryForLong( SQL );
```

A simple query using a bind variable

```
String SQL = "select age from Student where id = ?";
int age = jdbcTemplateObject.queryForInt(SQL, new Object[]{10});
```

Querying for a String

```
String SQL = "select name from Student where id = ?";
String name = jdbcTemplateObject.queryForObject(SQL, new Object[]{10}, String.class);
```

Querying and returning an object

```
String SQL = "select * from Student where id = ?";
Student student = jdbcTemplateObject.queryForObject(
  SQL, new Object[]{10}, new StudentMapper());

public class StudentMapper implements RowMapper<Student> {
  public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
    Student student = new Student();
    student.setID(rs.getInt("id"));
```

```
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));

      return student;
   }
}
```

Querying and returning multiple objects

```
String SQL = "select * from Student";
List<Student> students = jdbcTemplateObject.query(
   SQL, new StudentMapper());

public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
      Student student = new Student();
      student.setID(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));

      return student;
   }
}
```

Inserting a row into the table

```
String SQL = "insert into Student (name, age) values (?, ?)";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 11} );
```

Updating a row into the table

```
String SQL = "update Student set name = ? where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 10} );
```

Deleting a row from the table

```
String SQL = "delete Student where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{20} );
```

Executing DDL Statements

You can use the **execute(..)** method from *jdbcTemplate* to execute any SQL statements or DDL statements. Following is an example to use CREATE statement to create a table −

```
String SQL = "CREATE TABLE Student( " +
  "ID   INT NOT NULL AUTO_INCREMENT, " +
  "NAME VARCHAR(20) NOT NULL, " +
  "AGE  INT NOT NULL, " +
  "PRIMARY KEY (ID));"

jdbcTemplateObject.execute( SQL );
```

3.2 Week 7,8 Of Training

3.2.1 Junit

JUnit is a **Regression Testing Framework** used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit Framework can be easily integrated with either of the following −

- Eclipse
- Ant
- Maven

JUnit Parameterized Test

JUnit 4 has introduced a new feature called **parameterized tests**. Parameterized tests allow a developer to run the same test over and over again using different values. There are five steps that you need to follow to create a parameterized test.

- Annotate test class with @RunWith(Parameterized.class).
- Create a public static method annotated with @Parameters that returns a Collection of Objects (as Array) as test data set.

- Create a public constructor that takes in what is equivalent to one "row" of test data.
- Create an instance variable for each "column" of test data.
- Create your test case(s) using the instance variables as the source of the test data.

The test case will be invoked once for each row of data. Let us see parameterized tests in action.

Create a Class

Create a java class to be tested, say, **PrimeNumberChecker.java** in C:\>JUNIT_WORKSPACE.

```
public class PrimeNumberChecker {
  public Boolean validate(final Integer primeNumber) {
    for (int i = 2; i < (primeNumber / 2); i++) {
      if (primeNumber % i == 0) {
        return false;
      }
    }
    return true;
  }
}
```

Create Parameterized Test Case Class

Create a java test class, say, **PrimeNumberCheckerTest.java**. Create a java class file named **PrimeNumberCheckerTest.java** in C:\>JUNIT_WORKSPACE.

```
import java.util.Arrays;
import java.util.Collection;
import org.junit.Test;
import org.junit.Before;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import org.junit.runner.RunWith;
import static org.junit.Assert.assertEquals;

@RunWith(Parameterized.class)
public class PrimeNumberCheckerTest {
```

```
private Integer inputNumber;
private Boolean expectedResult;
private PrimeNumberChecker primeNumberChecker;

@Before
public void initialize() {
  primeNumberChecker = new PrimeNumberChecker();
}

// Each parameter should be placed as an argument here
// Every time runner triggers, it will pass the arguments
// from parameters we defined in primeNumbers() method

public PrimeNumberCheckerTest(Integer inputNumber, Boolean expectedResult) {
  this.inputNumber = inputNumber;
  this.expectedResult = expectedResult;
}

@Parameterized.Parameters
public static Collection primeNumbers() {
  return Arrays.asList(new Object[][] {
    { 2, true },
    { 6, false },
    { 19, true },
    { 22, false },
    { 23, true }
  });
}

// This test will run 4 times since we have 5 parameters defined
@Test
public void testPrimeNumberChecker() {
  System.out.println("Parameterized Number is : " + inputNumber);
  assertEquals(expectedResult,
  primeNumberChecker.validate(inputNumber));
```

```
  }
}
```

Create Test Runner Class

Create a java class file named **TestRunner.java** in C:\>JUNIT_WORKSPACE to execute test case(s).

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
  public static void main(String[] args) {
    Result result = JUnitCore.runClasses(PrimeNumberCheckerTest.class);

    for (Failure failure : result.getFailures()) {
      System.out.println(failure.toString());
    }

    System.out.println(result.wasSuccessful());
  }
}
```

Compile the PrimeNumberChecker, PrimeNumberCheckerTest and Test Runner classes using javac.

```
C:\JUNIT_WORKSPACE>javac PrimeNumberChecker.java PrimeNumberCheckerTest.java
TestRunner.java
```

Now run the Test Runner, which will run the test cases defined in the provided Test Case class.

```
C:\JUNIT_WORKSPACE>java TestRunner
```

Verify the output.

```
Parameterized Number is : 2
Parameterized Number is : 6
Parameterized Number is : 19
Parameterized Number is : 22
```

Parameterized Number is : 23
true

3.2.2 Mockito

What is Mocking?

Mocking is a way to test the functionality of a class in isolation. Mocking does not require a database connection or properties file read or file server read to test a functionality. Mock objects do the mocking of the real service. A mock object returns a dummy data corresponding to some dummy input passed to it.

Mockito

Mockito facilitates creating mock objects seamlessly. It uses Java Reflection in order to create mock objects for a given interface. Mock objects are nothing but proxy for actual implementations. Consider a case of Stock Service which returns the price details of a stock. During development, the actual stock service cannot be used to get real-time data. So we need a dummy implementation of the stock service. Mockito can do the same very easily, as its name suggests.

Consider the following code snippet.

```
import java.util.ArrayList;
import java.util.List;

import static org.mockito.Mockito.*;

public class PortfolioTester {
   public static void main(String[] args){

      //Create a portfolio object which is to be tested
      Portfolio portfolio = new Portfolio();

      //Creates a list of stocks to be added to the portfolio
      List<Stock> stocks = new ArrayList<Stock>();
      Stock googleStock = new Stock("1","Google", 10);
      Stock microsoftStock = new Stock("2","Microsoft",100);

      stocks.add(googleStock);
```

```
    stocks.add(microsoftStock);

    //Create the mock object of stock service
    StockService stockServiceMock = mock(StockService.class);

    // mock the behavior of stock service to return the value of various stocks
    when(stockServiceMock.getPrice(googleStock)).thenReturn(50.00);
    when(stockServiceMock.getPrice(microsoftStock)).thenReturn(1000.00);

    //add stocks to the portfolio
    portfolio.setStocks(stocks);

    //set the stockService to the portfolio
    portfolio.setStockService(stockServiceMock);

    double marketValue = portfolio.getMarketValue();

    //verify the market value to be
    //10*50.00 + 100* 1000.00 = 500.00 + 100000.00 = 100500
    System.out.println("Market value of the portfolio: "+ marketValue);
  }
}
```

Mockito with JUnit

we'll learn how to integrate JUnit and Mockito together. Here we will create a Math Application which uses CalculatorService to perform basic mathematical operations such as addition, subtraction, multiply, and division.

We'll use Mockito to mock the dummy implementation of CalculatorService. In addition, we've made extensive use of annotations to showcase their compatibility with both JUnit and Mockito.

The process is discussed below in a step-by-step manner.

Step 1 – Create an interface called CalculatorService to provide mathematical functions

File: CalculatorService.java

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

Step 2 – Create a JAVA class to represent MathApplication

File: MathApplication.java

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }

   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }

   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }

   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }

   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

Step 3 – Test the MathApplication class

Let's test the MathApplication class, by injecting in it a mock of calculatorService. Mock will be created by Mockito.

File: MathApplicationTester.java

```
import static org.mockito.Mockito.when;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;

// @RunWith attaches a runner with the test class to initialize the test data
@RunWith(MockitoJUnitRunner.class)
public class MathApplicationTester {

  //@InjectMocks annotation is used to create and inject the mock object
  @InjectMocks
  MathApplication mathApplication = new MathApplication();

  //@Mock annotation is used to create the mock object to be injected
  @Mock
  CalculatorService calcService;

  @Test
  public void testAdd(){
    //add the behavior of calc service to add two numbers
    when(calcService.add(10.0,20.0)).thenReturn(30.00);

    //test the add functionality
    Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);
  }
}
```

Step 4 – Create a class to execute to test cases

Create a java class file named TestRunner in **C> Mockito_WORKSPACE** to execute Test case(s).

File: TestRunner.java

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
  public static void main(String[] args) {
    Result result = JUnitCore.runClasses(MathApplicationTester.class);

    for (Failure failure : result.getFailures()) {
      System.out.println(failure.toString());
    }

    System.out.println(result.wasSuccessful());
  }
}
```

Step 5 – Verify the Result

Compile the classes using **javac** compiler as follows –

```
C:\Mockito_WORKSPACE>javac CalculatorService.java MathApplication.
  java MathApplicationTester.java TestRunner.java
```

Now run the Test Runner to see the result –

```
C:\Mockito_WORKSPACE>java TestRunner
```

Verify the output.

```
true
```

3.3 Week 9 Of Training:

3.3.1 Spring MVC

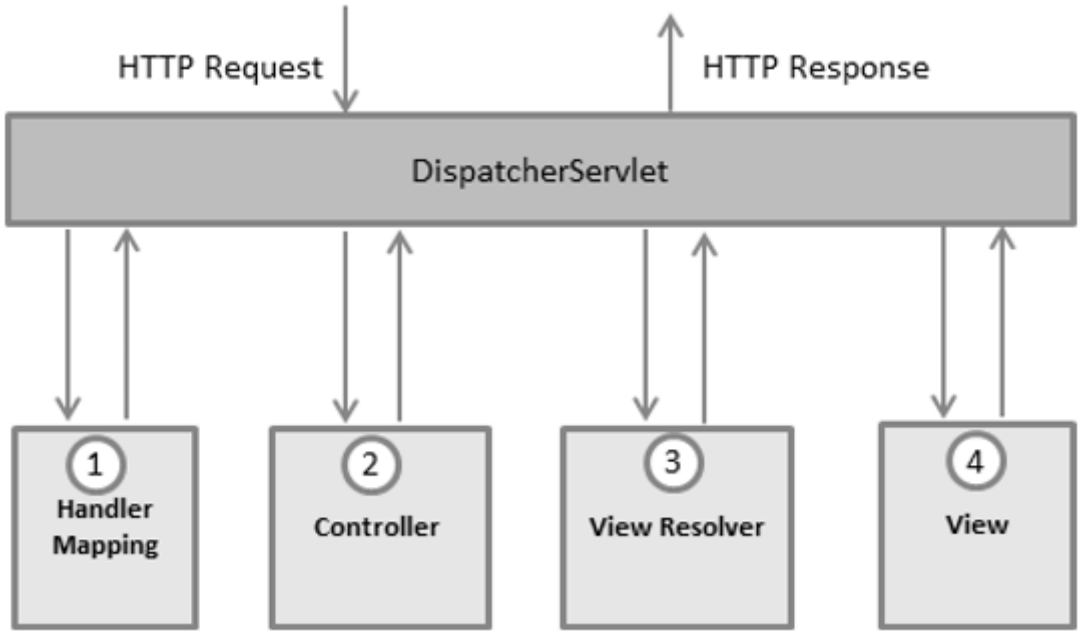
Spring MVC

The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

- The **Model** encapsulates the application data and in general they will consist of POJO.
- The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
- The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

The DispatcherServlet

The Spring Web model-view-controller (MVC) framework is designed around a *DispatcherServlet* that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC *DispatcherServlet* is illustrated in the following diagram –



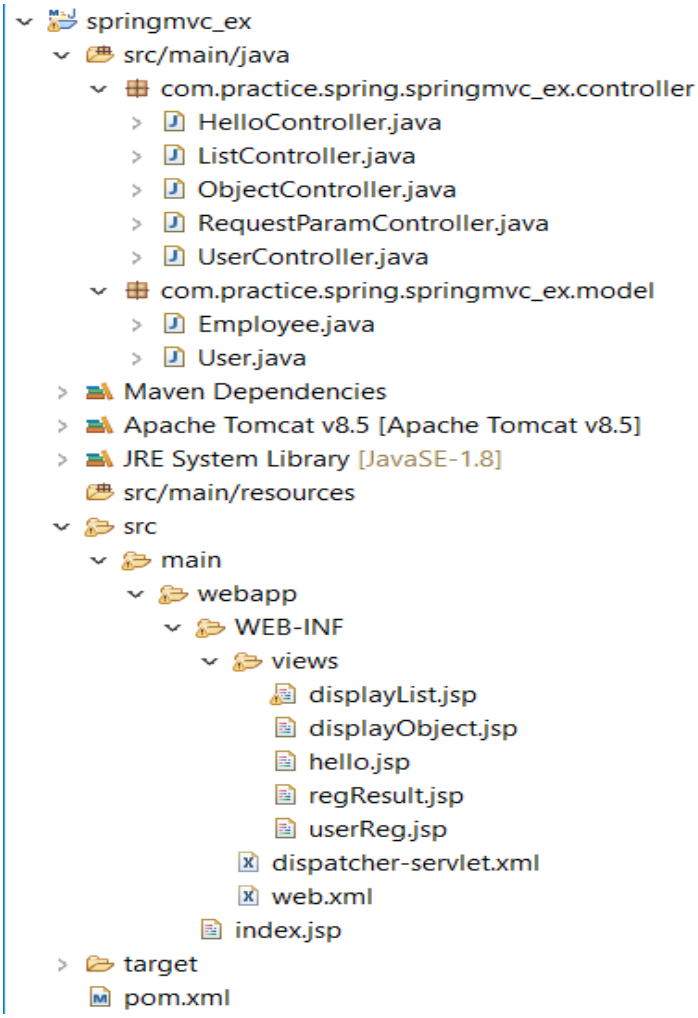
Following is the sequence of events corresponding to an incoming HTTP request to *DispatcherServlet* –

- After receiving an HTTP request, *DispatcherServlet* consults the *HandlerMapping* to call the appropriate *Controller*.

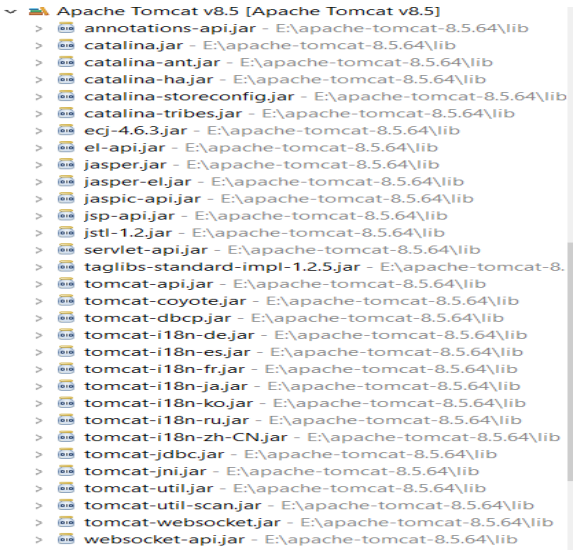
- The *Controller* takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
- The *DispatcherServlet* will take help from *ViewResolver* to pickup the defined view for the request.
- Once view is finalized, The *DispatcherServlet* passes the model data to the view which is finally rendered on the browser.

All the above-mentioned components, i.e. HandlerMapping, Controller, and ViewResolver are parts of *WebApplicationContext* w which is an extension of the plain*ApplicationContext* with some extra features necessary for web applications.

Hierarchy



Tomcat Server has to be configured



3.3.2 Spring Boot

What is Spring Boot?

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**. You can get started with minimum configurations without the need for an entire Spring configuration setup.

Why Spring Boot?

`You can choose Spring Boot because of the features and benefits it offers as given here –`

- It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- It provides a powerful batch processing and manages REST endpoints.
- In Spring Boot, everything is auto configured; no manual configurations are needed.
- It offers annotation-based spring application
- Eases dependency management
- It includes Embedded Servlet Container

How does it work?

Spring Boot automatically configures your application based on the dependencies you have added to the project by using **@EnableAutoConfiguration** annotation. For example, if

MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains **@SpringBootApplication** annotation and the main method.

Spring Boot automatically scans all the components included in the project by using **@ComponentScan** annotation.

Spring Boot Starters

Handling dependency management is a difficult task for big projects. Spring Boot resolves this problem by providing a set of dependencies for developers convenience.

For example, if you want to use Spring and JPA for database access, it is sufficient if you include **spring-boot-starter-data-jpa** dependency in your project.

Note that all Spring Boot starters follow the same naming pattern **spring-boot-starter-** *, where * indicates that it is a type of the application.

Examples

Look at the following Spring Boot starters explained below for a better understanding −

Spring Boot Starter Actuator dependency is used to monitor and manage your application. Its code is shown below −

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Spring Boot Starter Security dependency is used for Spring Security. Its code is shown below −

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Spring Boot Starter web dependency is used to write a Rest Endpoints. Its code is shown below −

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring Boot Starter Thyme Leaf dependency is used to create a web application. Its code is shown below –

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Spring Boot Starter Test dependency is used for writing Test cases. Its code is shown below –

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

Auto Configuration

Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

For this purpose, you need to add **@EnableAutoConfiguration** annotation or **@SpringBootApplication** annotation to your main class file. Then, your Spring Boot application will be automatically configured.

Observe the following code for a better understanding –

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@EnableAutoConfiguration
public class DemoApplication {
  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
```

```
  }
}
```

Spring Boot Application

The entry point of the Spring Boot Application is the class contains **@SpringBootApplication** annotation. This class should have the main method to run the Spring Boot application. **@SpringBootApplication** annotation includes Auto-Configuration, Component Scan, and Spring Boot Configuration.

If you added **@SpringBootApplication** annotation to the class, you do not need to add the **@EnableAutoConfiguration,**
@ComponentScan and **@SpringBootConfiguration** annotation.
The **@SpringBootApplication** annotation includes all other annotations.

Observe the following code for a better understanding –

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }
}
```

Component Scan

Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the **@ComponentScan** annotation for your class file to scan your components added in your project.

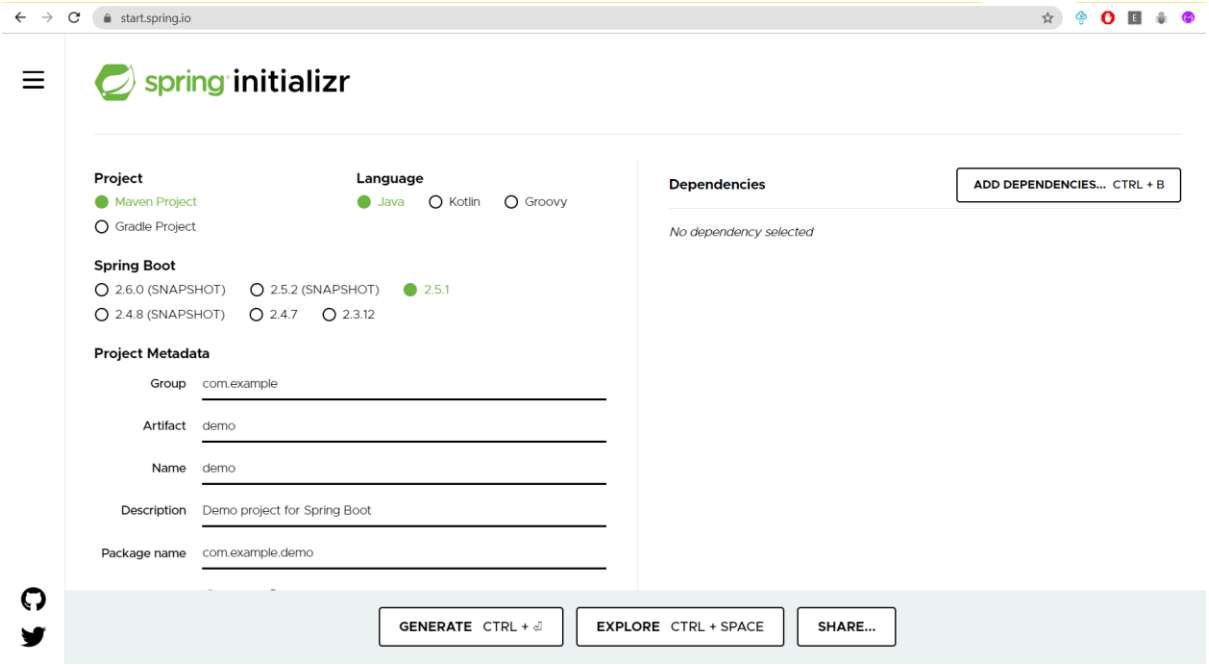
Observe the following code for a better understanding –

```
import org.springframework.boot.SpringApplication;
import org.springframework.context.annotation.ComponentScan;

@ComponentScan
public class DemoApplication {
  public static void main(String[] args) {
```

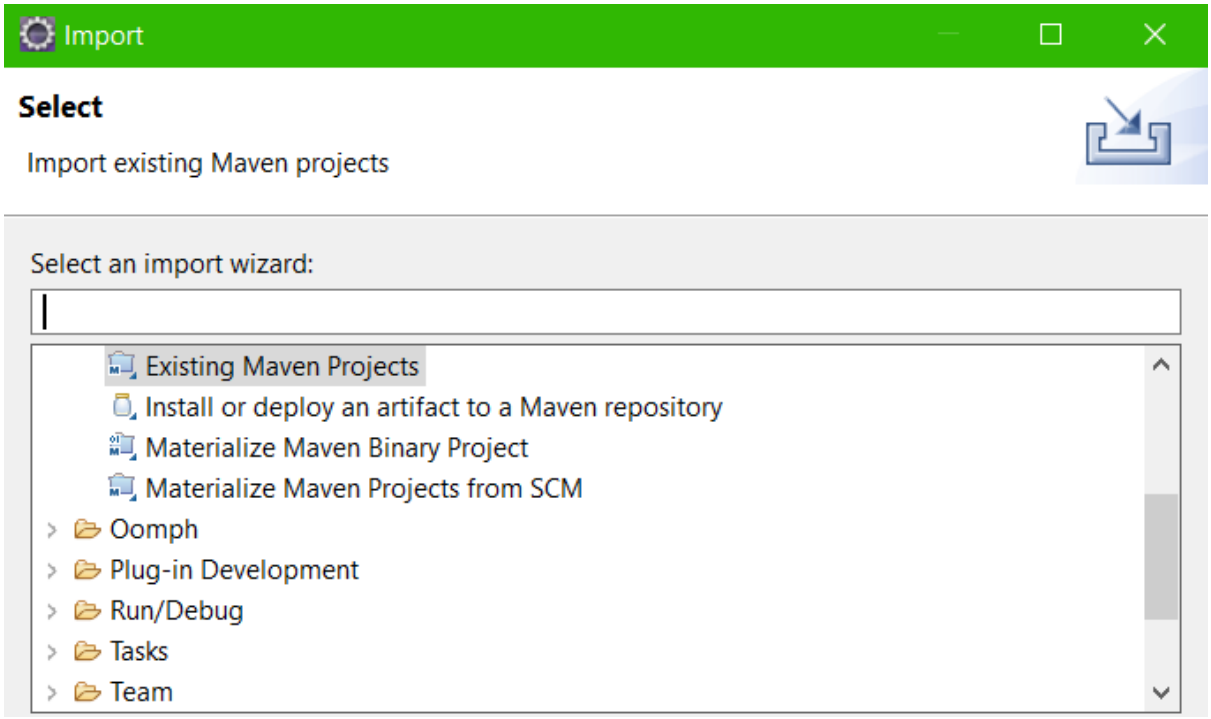
```
    SpringApplication.run(DemoApplication.class, args);
  }
}
```

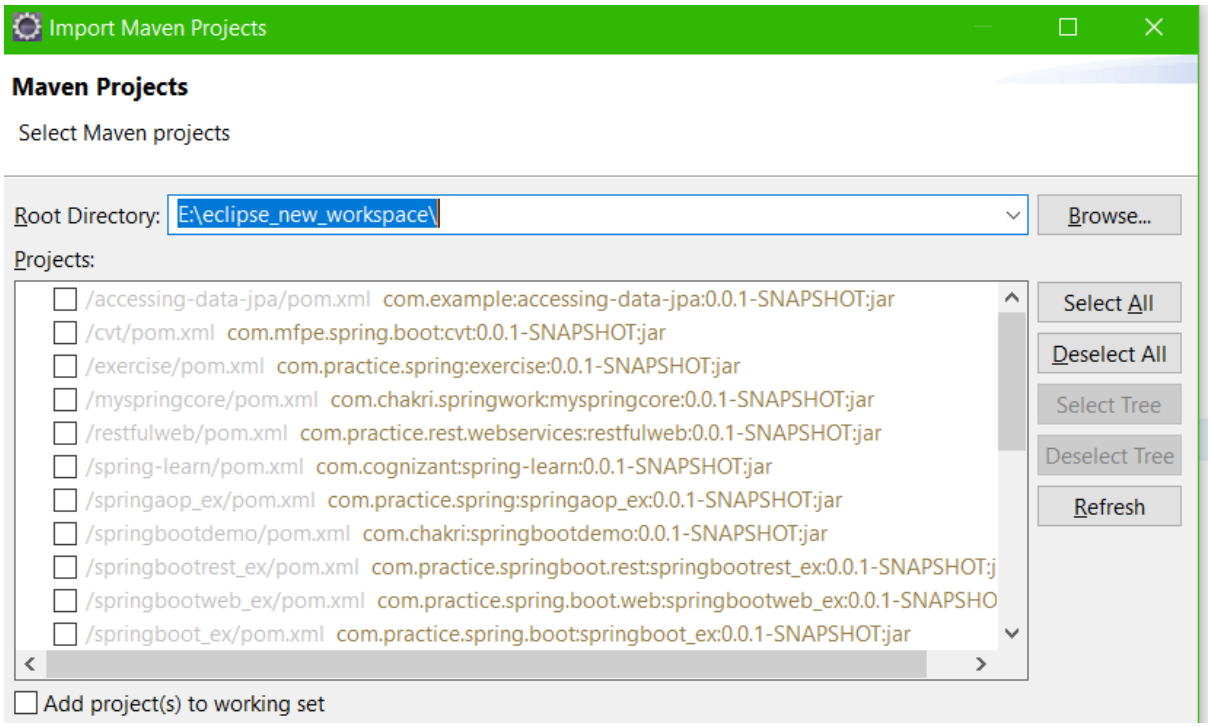
Creating a Spring Boot Maven Project using spring initializer



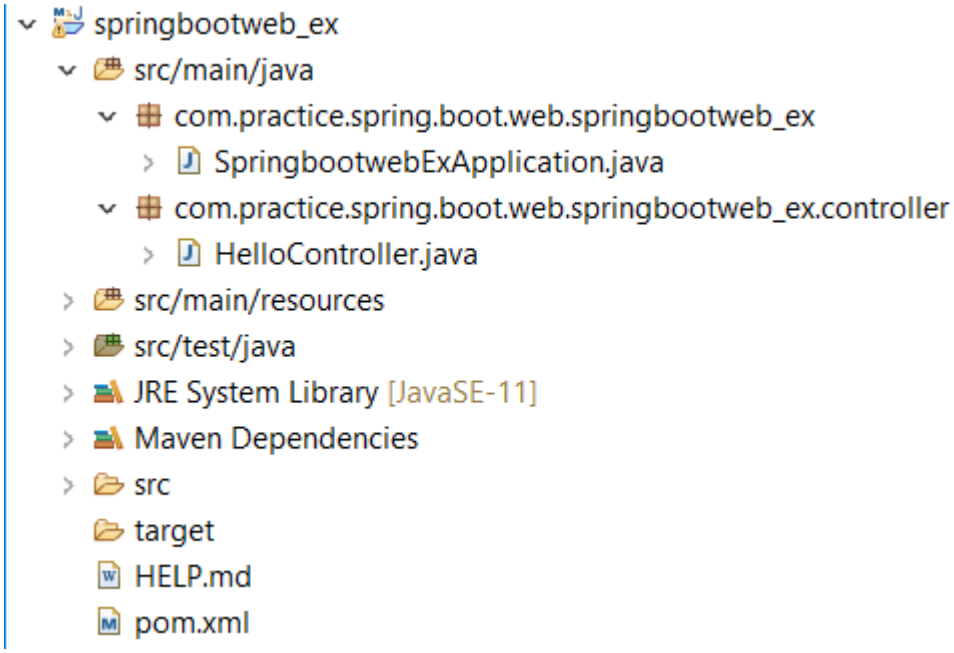
Extract the project into the eclipse directory using winRAR

Now in eclipse import the Maven project





After importing the project we can write the code by following the industry pattern.



The coding is similar to the one that we have written in spring mvc.

4.TECHNICAL TRAINING – STAGE 3

4.1 Week 10,11 Of Training

4.1.1 Spring Boot REST

Generally, the four primary HTTP verbs are used as follows:

GET

Read a specific resource (by an identifier) or a collection of resources.

PUT

Update a specific resource (by an identifier) or a collection of resources. Can also be used to create a specific resource if the resource identifier is known before-hand.

DELETE

Remove/delete a specific resource by an identifier.

POST

Create a new resource. Also a catch-all verb for operations that don't fit into the other categories.

Let's walk through this example. First, you can see that this controller class has a RequestMapping annotation at the class level (line 2). This is a common recommended practice which assigns responsibility for the endpoint ("/smartphones" in this case) to the class:

```
@RestController
@RequestMapping("/smartphones")
public class SmartphoneController {
```

Next, Spring Web MVC (it's actually part of Spring framework generally rather than specifically being part of Spring Boot) actually provides a mapping for each of the verbs as an annotation you can put on different methods. They all start with the HTTP verb (like Delete) and then the word mapping: PostMapping, PutMapping, GetMapping, DeleteMapping, etc. You can see the use of three of these on lines 10, 16, and 22 in this example.

When you annotate a method with one of these mappings, Spring will match up incoming requests to that method. Let's take the "addNewPhone" method here on line 11. When Spring sees an HTTP request come in with the POST verb and it was sent to the "/smartphones" endpoint, Spring resolves this method as the method responsible for it.

```
  @PostMapping
  public String addNewPhone(@RequestBody Smartphone smartphone) {
     phones.add(smartphone);
     return "OK";
  }
```

In the "addNewPhone" method, we take an HTTP request that is a POST request sent to the "/smartphones" endpoint, and look in the request body for JSON that matches the Smartphone class. The JSON looks like this:

```
{
  "brand": "Apple",
  "product": "iPhone"
}
```

In this example, we simply add it to a list of phones and then return the String value "OK."

The other methods similarly manage the list of phones. You can get the full list or delete a phone from the list. The video shows all of this in action and talks through it in more detail.

4.1.2 Git

Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows

4.1.3 JQuery

What is jQuery?

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto: **Write less, do more.** jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery –

- **DOM manipulation** – The jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- **Event handling** – The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support** – The jQuery helps you a lot to develop a responsive and feature rich site using AJAX technology.
- **Animations** – The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight** – The jQuery is very lightweight library – about 19KB in size (Minified and gzipped).
- **Cross Browser Support** – The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology** – The jQuery supports CSS3 selectors and basic XPath syntax.

4.2 Week 12 Of Training

4.2.1 Bootstrap

Bootstrap makes use of certain HTML elements and CSS properties that require the use of the HTML5 doctype. Hence include the below piece of code for HTML5 doctype at the beginning of all your projects using Bootstrap.

```
<!DOCTYPE html>
<html>
  ....
</html>
```

Mobile First

Since Bootstrap 3 has been launched, Bootstrap has become mobile first. It means 'mobile first' styles can be found throughout the entire library instead of them in separate files. You need to add the **viewport meta tag** to the **<head>** element, to ensure proper rendering and touch zooming on mobile devices.

```
<meta name = "viewport" content = "width = device-width, initial-scale = 1.0">
```

- *width* property controls the width of the device. Setting it to *device-width* will make sure that it is rendered across various devices (mobiles, desktops, tablets...) properly.
- *initial-scale = 1.0* ensures that when loaded, your web page will be rendered at a 1:1 scale, and no zooming will be applied out of the box.

Add **user-scalable = no** to the **content** attribute to disable zooming capabilities on mobile devices as shown below. Users are only able to scroll and not zoom with this change, and results in your site feeling a bit more like a native application.

```
<meta name = "viewport" content = "width = device-width, initial-scale = 1.0, maximum-scale = 1.0, user-scalable = no">
```

Normally *maximum-scale = 1.0* is used along with *user-scalable = no*. As mentioned above **user-scalable = no** may give users an experience more like a native app, hence Bootstrap doesn't recommend using this attribute.

Responsive Images

Bootstrap 3 allows you to make the images responsive by adding a class **.img-responsive** to the **<img>** tag. This class applies **max-width: 100%;** and **height: auto;** to the image so that it scales nicely to the parent element.

<img src = "..." class = "img-responsive" alt = "Responsive image">

Typography and Links

Bootstrap sets a basic global display (background), typography, and link styles –

- **Basic Global display** – Sets *background-color: #fff;* on the <*body*> element.
- **Typography** – Uses the *@font-family-base, @font-size-base,* and *@line-height-base* attributes as the typographic base.
- **Link styles** – Sets the global link color via attribute *@link-color* and apply link underlines only on *:hover*.

4.2.2 React

ReactJS is a JavaScript library used for building reusable UI components. According to React official documentation, following is the definition –

React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding.

React Features

- **JSX** – JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
- **Components** – React is all about components. You need to think of everything as a component. This will help you maintain the code when working on larger scale projects.
- **Unidirectional data flow and Flux** – React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.
- **License** – React is licensed under the Facebook Inc. Documentation is licensed under CC BY 4.0.

Using JSX

JSX looks like regular HTML in most cases. We already used it in the Environment Setup chapter. Look at the code from **App.jsx** where we are returning **div**.

App.jsx

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        Hello World!!!
      </div>
    );
  }
}
export default App;
```

Even though it's similar to HTML, there are a couple of things we need to keep in mind when working with JSX.

Nested Elements

If we want to return more elements, we need to wrap it with one container element. Notice how we are using **div** as a wrapper for **h1**, **h2** and **p** elements.

App.jsx

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
```

```
          <h2>Content</h2>
          <p>This is the content!!!</p>
        </div>
      );
    }
}
export default App;
```

Attributes

We can use our own custom attributes in addition to regular HTML properties and attributes. When we want to add custom attribute, we need to use **data-** prefix. In the following example, we added **data-myattribute** as an attribute of **p** element.

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
        <h2>Content</h2>
        <p data-myattribute = "somevalue">This is the content!!!</p>
      </div>
    );
  }
}
export default App;
```

JavaScript Expressions

JavaScript expressions can be used inside of JSX. We just need to wrap it with curly brackets **{}**. The following example will render **2**.

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{1+1}</h1>
      </div>
    );
  }
}
export default App;
```

We cannot use **if else** statements inside JSX, instead we can use **conditional (ternary)** expressions. In the following example, variable **i** equals to **1** so the browser will render **true**, If we change it to some other value, it will render **false**.

```
import React from 'react';

class App extends React.Component {
  render() {
    var i = 1;
    return (
      <div>
        <h1>{i == 1 ? 'True!' : 'False'}</h1>
```

```
        </div>
      );
    }
}
export default App;
```

Styling

React recommends using inline styles. When we want to set inline styles, we need to use **camelCase** syntax. React will also automatically append **px** after the number value on specific elements. The following example shows how to add **myStyle** inline to **h1** element.

```
import React from 'react';

class App extends React.Component {
  render() {
    var myStyle = {
      fontSize: 100,
      color: '#FF0000'
    }
    return (
      <div>
        <h1 style = {myStyle}>Header</h1>
      </div>
    );
  }
}
export default App;
```

Comments

When writing comments, we need to put curly brackets **{}** when we want to write comment within children section of a tag. It is a good practice to always use **{}** when writing comments, since we want to be consistent when writing the app.

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
        {//End of the line Comment...}
        {/*Multi line comment...*/}
      </div>
    );
  }
}
export default App;
```

Naming Convention

HTML tags always use **lowercase** tag names, while React components start with **Uppercase**.

Note – `You should use` **className** and **htmlFor** as XML attribute names instead of **class** and **for**.

`This is explained on React official page as –`

Since JSX is JavaScript, identifiers such as **class** and **for** are discouraged as XML attribute names. Instead, React DOM components expect DOM property names such as **className** and **htmlFor**, respectively.

There are two types of components in React:

1. Functional Component
2. Class Component

Functional Component:

A **Functional component** is a **function** that takes props and returns JSX. They do not have state or lifecycle methods. **Functional components** are easier to read, debug, and test. They offer performance benefits, decreased coupling, and greater reusability.

Class Component:

When creating a React component, the component's name must start with an uppercase letter.The component has to include the extended React.Component statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render() method, this method returns HTML.

4.3 Week 13 Of Training

4.3.1 Microservices

Microservice is a service-based application development methodology. In this methodology, big applications will be divided into smallest independent service units. Microservice is the process of implementing Service-oriented Architecture (SOA) by dividing the entire application as a collection of interconnected services, where each service will serve only one business need.

The Concept of Going Micro

In a service-oriented architecture, entire software packages will be subdivided into small, interconnected business units. Each of these small business units will communicate to each

other using different protocols to deliver successful business to the client. Now the question is, how Microservice Architecture (MSA) differs from SOA? In one word, SOA is a designing pattern and Microservice is an implementation methodology to implement SOA or we can say Microservice is a type of SOA.

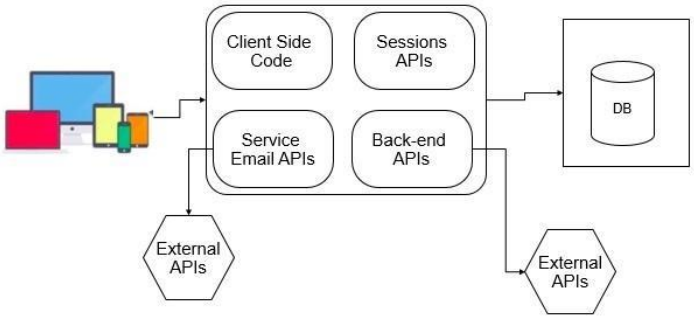
Following are some rules that we need to keep in mind while developing a Microservice-oriented application.

- **Independent** – Each microservice should be independently deployable.
- **Coupling** – All microservices should be loosely coupled with one another such that changes in one will not affect the other.
- **Business Goal** – Each service unit of the entire application should be the smallest and capable of delivering one specific business goal.

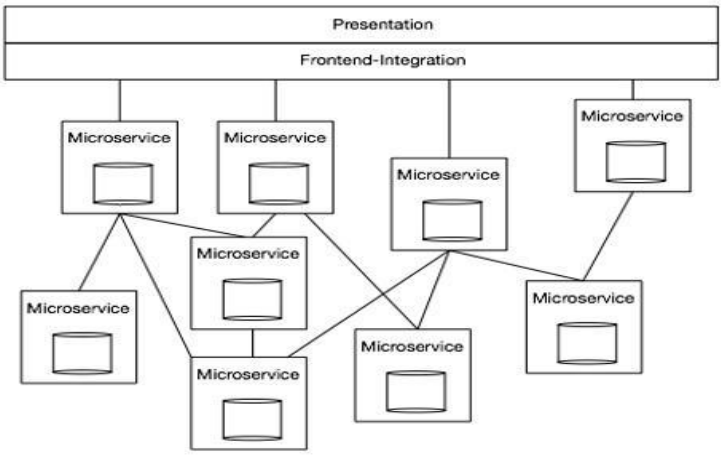
Let us consider an example of an online shopping portal to understand microservice in depth. Now, let us break this entire E-commerce portal into small business units such as user management, order management, check-in, payment management, delivery management, etc. One successful order needs to proceed through all of these modules within a specific time frame. Following is the consolidated image of different business units associated with one electronic commerce system.

Each of these business modules should have its own business logic and stakeholders. They communicate with other third party vendor softwares for some specific needs, and also with each other. For example, order management may communicate with user management to get user information.

Now, considering you are running an online shopping portal with all of these business units mentioned earlier, you do need some enterprise level application consisting of different layers such as front-end, back-end, database, etc. If your application is not scaled and completely developed in one single war file, then it will be called a typical monolithic application. According to IBM, a typical monolithic application should possess the following module structure internally where only one endpoint or application will be responsible to handle all user requests.



In the above image, you can see different modules such as Database for storing different users and business data. At the front-end, we have different devices where we usually render user or business data to use. In the middle, we have one package that can be a deployable EAR or WAR file that accepts requests from the users end, processes it with the help of the resources, and renders it back to the users. Everything will be fine until business wants any changes in the above example. Consider the following scenarios where you have to change your application according to the business needs. Business unit needs some changes in the "Search" module. Then, you need to change the entire search process and redeploy your application. In that case, you are redeploying your other units without any changes at all.



Now again your business unit needs some changes in the “Check out” module to include the “wallet” option. You now have to change your “Check out” module and redeploy the same into the server. Note, you are redeploying the different modules of your software packages, whereas we have not made any changes to it. Here comes the concept of service-oriented architecture more specific to Microservice architecture. We can develop our monolithic application in such a manner that each and every module of the software will behave as an independent unit, capable of handling a single business task independently. Consider the following example.

In the above architecture, we are not creating any ear file with compact end-to-end service. Instead, we are dividing different parts of the software by exposing them as a service. Any part of the software can easily communicate with each other by consuming respective services. That's how microservice plays a great role in modern web applications.

Let us compare our shopping cart example in the line of microservice. We can break down our shopping cart into different modules such as “Search”, ”Filter”, “Checkout”, “Cart”, “Recommendation”, etc. If we want to build a shopping cart portal then we have to build the above-mentioned modules in such a manner that they can connect to each other to give you a 24x7 good shopping experience.

Microservices Over SOA

The following table lists certain features of SOA and Microservice, bringing out the importance of using microservice over SOA.

Component	SOA	Microservice
Design pattern	SOA is a design paradigm for computer software, where software components are exposed to the outer world for usage in the form of services.	Micro Service is a part of SOA. It is a specialized implementation of SOA.

Dependency	Business units are dependent on each other.	All business units are independent of each other.
Size	Software size is bigger than conventional software.	Software size is small.
Technology	Technology stack is less than Microservice.	Microservice is heterogeneous in nature as exact technologies are used to perform a specific task. Microservices can be considered as a conglomerate of many technologies.
Autonomous and Focus	SOA applications are built to perform multiple business tasks.	Microservice applications are built to perform a single business task.
Nature	Monolithic in nature.	Full stack in nature.
Deployment	Deployment is time-consuming.	Deployment is very easy. Hence, it will be less time-consuming.
Cost-effectiveness	More cost-effective.	Less cost-effective.
Scalability	Less compared to Microservices.	Fully scaled.

4.3.2 AWS

What is Cloud Computing?

Cloud computing is an internet-based computing service in which large groups of remote servers are networked to allow centralized data storage, and online access to computer services

or resources. Using cloud computing, organizations can use shared computing and storage resources rather than building, operating, and improving infrastructure on their own.

Cloud computing is a model that enables the following features.

- Users can provision and release resources on-demand.
- Resources can be scaled up or down automatically, depending on the load.
- Resources are accessible over a network with proper security.
- Cloud service providers can enable a pay-as-you-go model, where customers are charged based on the type of resources and per usage.

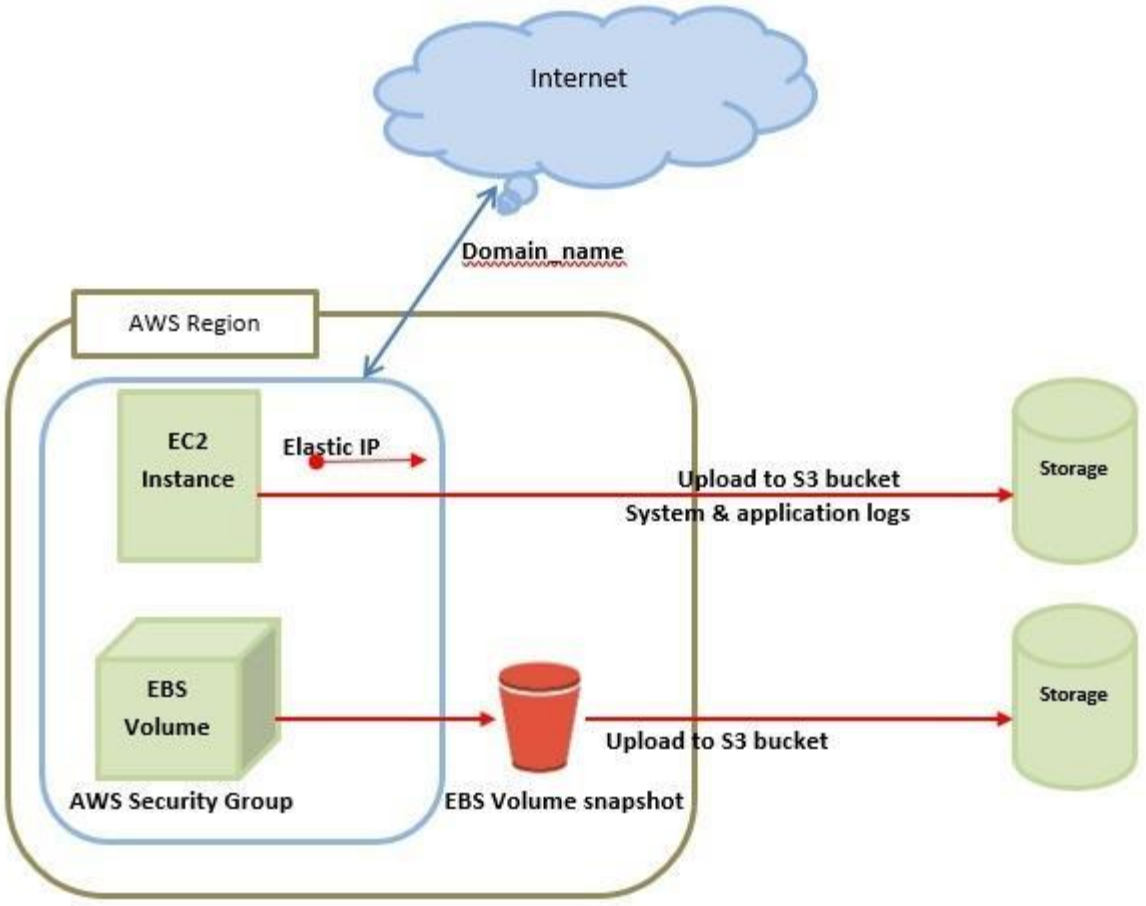
Types of Clouds

There are three types of clouds – Public, Private, and Hybrid clouds.

- Public Cloud
- Private Cloud
- Hybrid Cloud

AWS Architecture

This is the basic structure of **AWS EC2**, where **EC2** stands for Elastic Compute Cloud. EC2 allows users to use virtual machines of different configurations as per their requirement. It allows various configuration options, mapping of individual servers, various pricing options, etc. We will discuss these in detail in the AWS Products section. Following is the diagrammatic representation of the architecture.



6. CONCLUSION

I can say that my time spent interning with COGNIZANT(CTS) resulted in one of the best hands on learning experiences I have gained. Not only did I gain a technical skill set, but I also had the opportunity to shape my soft skills. The atmosphere set my manager always welcoming and provided the right learning environment. Additionally, I felt like I was able to contribute to the company by assisting and working on various skills throughout the Internship period. The internship was a very useful experience as I have found out what my strengths and weaknesses are. I gained new knowledge and skills and met many new people. I achieved many of my learning goals in the analytics fields. These projects can serve as a foundation stone for other integration projects in the organization. The Analytics space is constantly evolving, and the pace of transformation is only accelerating as expectations increase for streamlined and transparent experiences, and IT environments grow more complex to support business initiatives.

7. REFERENCES

[1] https://cognizantlearn.cognizant.com/

[2] https://www.javaguides.net/2020/07/react-js-spring-boot-rest-api-example-tutorial.html

[3] https://github.com/axios/axios

[4] https://spring.io/guides/gs/accessing-data-jpa/

[5] https://www.baeldung.com/spring-mvc

[6] https://www.journaldev.com/21127/spring-boot-rest-example

[7] https://github.com/in28minutes/full-stack-with-react-and-spring-boot

[8] https://www.w3schools.com/jquery/

[9] https://getbootstrap.com/docs/5.0/getting-started/introduction/

[10] https://getbootstrap.com/docs/5.0/content/tables/

[11] https://reactjs.org/docs/components-and-props.html

[12] https://github.com/nandinishamdasani77?tab=repositories

[13] https://start.spring.io/

[14] https://aws.amazon.com/console