JAVA PROGRAMMING Simplified

From Novice to Professional - Start at the Beginning and Learn the World of Java

DR. MUNEER AHMAD DAR

bpb

JAVA PROGRAMMING Simplified

From Novice to Professional - Start at the Beginning and Learn the World of Java

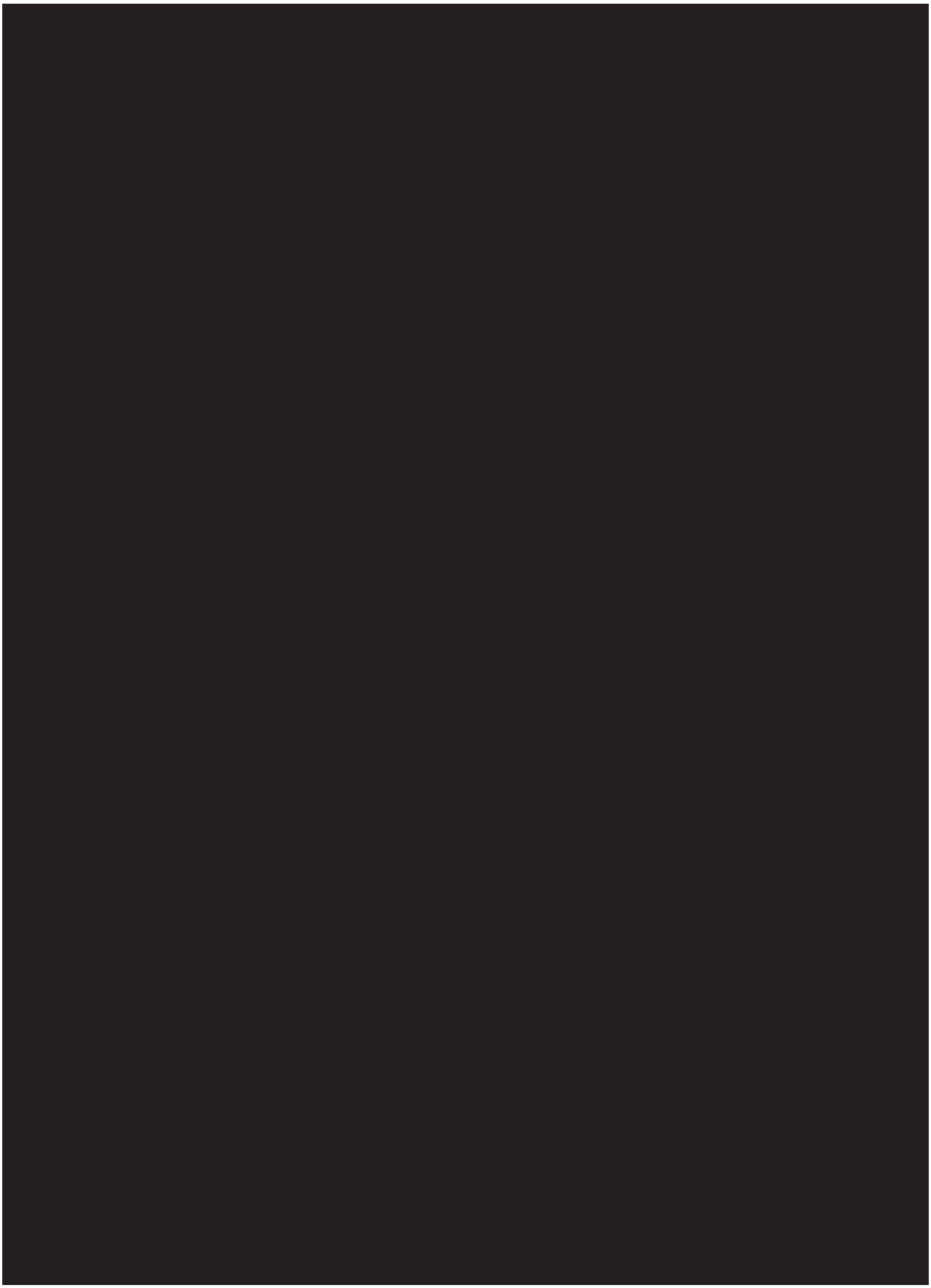
DR. MUNEER AHMAD DAR

bpb

Java

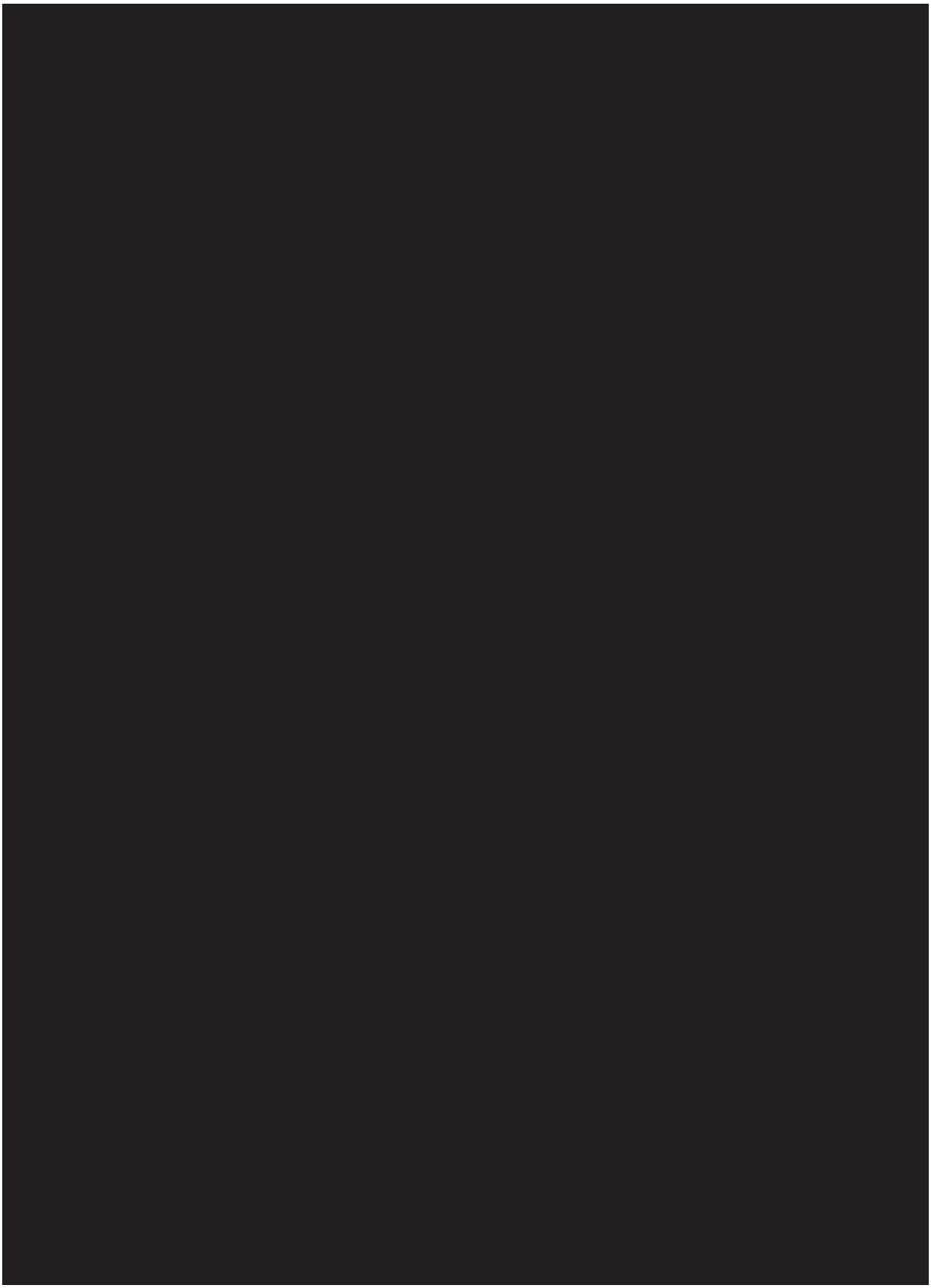
Programming

Simplified



From Novice to Professional - Start at the

Beginning and Learn the World of Java



by

Dr. Muneer Ahmad Dar

FIRST EDITION 2020

ISBN: 978-93-89845-143

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author’s & publisher’s knowledge. The author has made every effort to ensure the accuracy of these publications, but cannot be held responsible for any loss or damage arising from any information in this book.

Distributors:

BPB PUBLICATIONS

20, Ansari Road, Darya Ganj

New Delhi-110002

Ph: 23254990/23254991

MICRO MEDIA

Shop No. 5, Mahendra Chambers,

150 DN Rd. Next to Capital Cinema,

V.T. (C.S.T.) Station, MUMBAI-400 001

Ph: 22078296/22078297

DECCAN AGENCIES

4-3-329, Bank Street,

Hyderabad-500195

Ph: 24756967/24756400

BPB BOOK CENTRE

376 Old Lajpat Rai Market,

Delhi-110006

Ph: 23861747

Published by Manish Jain for BPB Publications, 20 Ansari Road, Darya Ganj, New Delhi-110002 and Printed by him at Repro India Ltd, Mumbai

Dedicated to

My loving daughters

Maheen Muneer and Fajr Muneer

About the Author

Dr. Muneer Ahmad Dar is currently working as Scientist-C at National Institute of Electronics and Information Technology (NIELIT), J&K, which is the department under Ministry of Electronics and Information Technology, MeitY, Govt of India. He is a researcher, teacher and Head, Department of MCA at NIELIT Srinagar. He is actively involved in the field of Computer Science. He has done his Masters in Computer Applications (MCA) from University of Kashmir, M.Phil. (Computer Science) from Madurai Kamaraj University, and PhD (Computer Science) from University of Kashmir. His areas of interest include security of smartphone applications, programming languages, design and analysis of algorithms, data structures, and optimization techniques. As a creative writer, he has authored a large number of research papers and book chapters, published in IEEE, Scopus indexed journals and Springer lecture notes.

About the Reviewer

Syed Nisar Bukhari is working as Scientist-Cin National Institute of Electronics and Information Technology (NIELIT) J&K Srinagar, an institute under Ministry of Electronics and Information Technology, Govt. of India. He is also Head department of Information Technology (IT) at NIELIT Srinagar. He has 12 years of experience in Research, Development and Academics. Before joining NIELIT Syed Nisar Bukhari was working as Software Engineer wherein he was involved in developing complex enterprise level windows, web and service based applications using latest technologies. His area of research is Machine Learning and Deep Learning and his problem area is Immunoinformatics (Immunology) which fall under bio informatics wherein he is applying machine learning techniques on biological databases to predict the results. He is also into web development and is working on the security aspects of web applications. Syed Nisar Bukhari has also published number of papers in various national and International journals and conferences.

Acknowledgement

Almighty Allah – the most beneficial and the merciful, the lord of infinite matter in the universe. I am thankful to my merciful lord Allah for his innumerable and gracious blessings on me during my whole life period. His countless blessings boosted and motivated me to write this book. Thank You, O Allah. Besides this, I owe splendid thanks to many people who helped, motivated and supported me during the writing of this book. The credit of successful completion of this book depends largely on the encouragement and persuasion of my parents and my wife. As writing the book is a time consuming process so I would like to thank my two little daughters whose time I have utilized and not spent much of the time with them. I would like to take this opportunity to express my sincere gratitude to the people who have been directly or indirectly involved in the successful completion of this book.

I also acknowledge the support from the technical and non-technical staff of NIELIT Srinagar and thank them for their immense support.

I also sincerely thank all my students who have motivated me to write this book in the easiest possible way so that they can study at home during the unfavorable conditions.

Last but not least, I would like to thank my brother without whose love, support and blessing, this book would not have been possible.

Preface

This book has been designed in such a manner to make anyone understand the Java language with lot of practical examples implemented on the Eclipse platform. This book comprehensively covers all the concepts of Java, staring with the installation of Java and usage of IDE for Java development. This book moves further by elaborating the basic constructs of Java language and further moves on to explain the object oriented concepts of Java. All major topics like Exception Handling, Packages and Interfaces, Applets, Event Handling and Java Database Connectivity are explained with lot of relevant programs.

The motivation behind writing this book on Java is the hardships of my beloved students who are not able to attend the college regularly because of the unavoidable circumstances. This book provides a best platform for those students who want to self-study without the instructor as most of the programs are covered in this book.

This book has provided a detailed description about the topic to first understand it and then comprehensive programs are implemented to have more clarity on the topic. This book is well suited for the first level programming course on Javafor the students of BCA, BSc-IT, and MSc in Computer Science and Information Technology, B. Tech, and BE in Computer Science and Information Technology. It would be a good book for the students of MCA, MSc-IT, and NIELIT courses.

In conclusion, this book is for all the passionate programmers and students who want to learn Java from the scratch and want to implement some advanced projects in Java.

About the Book

This book systematically tries to cover all the concepts of Java programming language. The main objective of this book is to provide the beginner programmers with a platform to understand the popularity of Java and explains the users the importance of Java. This book is useful for beginner programmers having no knowledge of any programming language however, programmers who have done some basic programming in C and C++ can easily reach to some advanced concepts and move ahead with the advanced Java. The main focus of this book is to cover maximum number of programs with the stress on hands on sessions. The book is organized as under:

Chapter 1: Provides the detailed introduction about the Java language including the features of Java. It further explains the step-by-step installation of Java.

Chapter 2: After installing the Java programming language, the main objective of this chapter is to understand the basics of Java programming. The structure of the Java program is explained and the control structures are introduced in this chapter.

Chapter 3: The object-oriented programming is introduced in this chapter. The concepts like classes and objects are explained with some excellent programming examples.

Chapter 4: The packages and interfaces are explained in this chapter to get the insight of the built in Java packages.

Chapter 5: The arrays are explained in this chapter to get the insight about the one-dimensional and two-dimensional arrays. The strings handling is considered as a major task in implementing any Java project. This chapter explains the implementation of string handling with the help of String and StringBuffer class. The Wrapper classes are also explained in this chapter.

Chapter 6: It is important for a developer to understand the working of exception handling. This chapter explains the concept of exception handling. The user-defined exceptions are created to fully understand the working of try, catch, and finally blocks.

Chapter 7: One of the important capabilities of Java is the multithreading. The multiple threads are created through relevant Java examples and working of threads is explained in this chapter. How the threads synchronize and communicate with each other is also explained in this chapter.

Chapter 8: The applets, which are Java programs and can be executed on the internet, are introduced in this chapter.

Chapter 9: The input and output streams are explained in this chapter to retrieve the data from the files and other sources.

Chapter 10: As the graphical user, interfaces are created by the developers to interact with the user. This interaction is only possible through event handling. The delegation event model is explained in this chapter.

Chapter 11: The Java database connectivity is explained with the use of java.sql package. The JDBC architecture is explained to understand the various classes to be used to access the data in a database.

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors if any, occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Table of Contents

2. Basics of Java Programming

3. Object-Oriented Programming (OOP) in Java

9. Input and Output in Java

10. Event Handling in Java

Connection

Statement

ResultSet

Assignments

CHAPTER 1

Introduction and Installation

Introduction

Every enterprise uses Java in one-way or other. As per Oracle, more than 3 billion devices run applications designed on this development platform. Java is used to design the applications ranging from desktop GUI, embedded systems, web applications, including e-commerce applications, front and back office electronic trading systems, settlement and confirmation systems, data processing, web servers and application servers, mobile applications including Android applications, enterprise applications, scientific applications, and middleware products.

Having the wide range of applications we need enough individuals with skills in Java application development using Java to develop this infrastructure. Programming languages grow for two reasons: to become accustomed to transformations in surroundings and to put into practice advancements in the ability of programming. The environmental alteration that encouraged Java was the requirement for platform-independent programs intended for sharing on the internet.

On the other hand, Java also embodies changes in the way that people move towards the writing of programs. for instance, Java improved and sophisticated the object-oriented model used by C++ and other object oriented programming languages, furtherincorporated support for multithreading, and provided a library that simplified internet access. Java is the idealreply to the demands of therecentlybudding, extremelydispersed computing world. Java is to Internet programming what C was to system programming: aninnovativestrength that changed the world.

Structure

Introduction

History of Java

Features of Java

Downloading and installing the JDK

JDK version

Configuring Java development environment

The Eclipse IDE

Assignments

Objective

This chapter provides a detailed background of the java programming language. After reading this chapter, a reader will be capable of installing the Java and handling the Eclipse IDE to design and develop various Java projects.

History of Java

Initially named as Oak in 1991 by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan was later renamed as Java in 1995 while working at Sun MicroSystems, many people contributed to the blueprint and advancement of Java language but the people who contributed mainly are Tim Lindholm, Frank Yellin, Jonathan Payne, Bill Joy, and Arthur van Hoff.The novel thrust meant for Java was not the internet based applications but, the principal inspiration was the requirement for a platform-independent or in other words the architecture-neutral language that may perhaps be used to generate software to be embedded in a variety of customer electronic equipments, for instance microwave ovens and remote controls. As we can most likely presume, many different types of CPUs are used as controllers. In an effort to discover such a resolution, Gosling and others began work on a portable, platform-independent language that may well be used to construct code that would run on a range of CPUs under differing platforms. This endeavoreventually led to the foundation of Java programming language.

Features of Java

The popularity of Java is because of its wide range of features that contributed to the success of Java as a language. Some of the features of Java are as under:

Simple and familiar: With the vision of Sun Microsystems to develop a language, which is simple and familiar, the Java language has its syntax from C and object-oriented concepts from C++. Java being a case sensitive language as is the case of C, it has removed the concept of pointers (except this pointer), operator overloading, template classes, global variables, and header files. A programmer having good knowledge of C and C++ can make use of the syntax and control structures from C and object-oriented concepts from C++. This feature of Java makes it a simple and familiar language.

Compiled and interpreted: If we consider the case of any other language, it is either compiled or interpreted as in C we compile it and run it to generate the output. In Java we first compile it using javac (Compiler) to generate the bytecode file having .class extension. After compilation we have to use java (Interpreter) to generate the output. This two-step process that generates the bytecode file makes this language secure and platform independent.

Platform independent: The main reason for the success of Java programming language is its capability to run any platform. Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating system, processors, and system resources will not force any changes in the Java program. This is the reason for Java being so popular for the internet programming. One such example

is the use of Java applets as they can be downloaded from a remote computer onto our local machine through internet and can be executed locally.

As already explained, Java programs are compiled, a bytecode file is generated, and this file can be executed on any machine. The second advantage of using Java is the size of its primitive data types, which are machine independent.

Object-oriented: The most important feature of using Java as a programming language is its capability to encapsulate and implement everything in terms of objects and classes. Java being a true object-oriented language has a vast variety of packages (group of classes) to implement a wide range of functionalities. Thus, we may say that whatever the advantages and features of an object-oriented programming are, they are automatically applicable to Java.

Secure and robust: Being implemented as a first choice language for the internet programming, the Java has secured itself from number of security risks. The absence of pointers, verification of all memory access and proper scrutiny of applets are some of the measures taken by Java to make it secure.

Java has a strict compile time and run time checking for data types to ensure the reliable code. The exception handling that captures the run time errors and eliminates the risk of crashing the system and proper memory management in terms of garbage collection are the reasons for this language being robust.

Multi threaded: Java language is capable of dividing its program into multiple tasks and executing them simultaneously. For example, one task may be copying the file while another two tasks are displaying an animation

and playing the music. The user is given the impression that all the three tasks are getting executed simultaneously. The operating system issues like the inter thread communication and synchronization is efficiently implemented by themultithreading capability of Java.

Performance: The performance of Java is high mainly due to the intermediate bytecode file and its efficient architecture to reduce overheads during runtime. Furthermore, the multithreading contributes to the Java's performance by enhancing the overall execution speed of Java programs.

Programmer friendly: The features like enhanced forloop, generics, autoboxing or unboxing, type safeenums, static import and annotations make Java a programmer friendly language.

There are many other features like scalability and performance, JDBC, core XML support, and many more, on the basis of which Java is considered as a powerful and popular language.

Downloading and installing the JDK

The Java Development Kit (JDK), named officially as Java Platform Standard Edition or Java SE, is the main requirement for writing Java programs ranging from small to a large java projects. The JDK is freely available from Sun Microsystems (now part of Oracle). The JDK (Java SE) can be downloaded free of cost from the below website.
http://www.oracle.com/technetwork/java/javase/overview/index.html.

JDK, which includes Java Runtime (JRE) and the various development executables (such as compiler and debugger), is a mandatory requirement for writing Java projects and further executing the Java programs. JRE is needed for running Java programs. In other words, JRE is a subset of JDK. As we are going to write and execute the java projects, we should install JDK, which includes the java runtime environment.

JDK version

Right from the inception of Java language, the different versions of Java are produced to enhance its capability and provide the programmer with new packages so as to implement some major projects in Java. The following versions of Java are already released. Reference: https://en.wikipedia.org/wiki/Java_version_history.

JDK Alpha and Beta (1995): Sun Microsystem announced Java in September 23, 1995.

JDK 1.0 (January 1996): Originally called Oak (named after the oak tree outside James Gosling's office). Renamed to Java 1 in JDK 1.0.2.

JDK 1.1 (February 1997): Introduced AWT event model, inner class, JavaBean, JDBC, and RMI.

J2SE 1.2 (JDK 1.2) (December 1998): Re-branded as Java 2 and renamed JDK to J2SE (Java 2 Standard Edition). Also released J2EE (Java 2 Enterprise Edition) and J2ME (Java 2 Micro Edition). Included JFC (Java Foundation Classes - Swing, Accessibility API, Java 2D, Pluggable Look & Feel, and Drag & Drop). Also introduced Collection Framework and JIT compiler.

J2SE 1.3 (JDK 1.3) (May 2000): Introduced Hotspot JVM.

J2SE 1.4 (JDK 1.4) (February 2002): Introduced assert statement, non-blocking IO (nio), logging API, image IO, Java webstart, regular expression (regex) support.

J2SE 5.0 (JDK 5) (September 2004): Officially called 5.0 instead of 1.5. Introduced generics, autoboxing/unboxing, annotation, enum, varargs, for-each loop, static import.

Java SE 6 (JDK 6) (December 2006): Renamed J2SE to Java SE.

Java SE 7 (JDK 7) (July 2011): First version after Oracle purchased Sun Microsystem - also called Oracle JDK.

Java SE 8 (JDK 8) (LTS) (March 2014): Included support for Lambda expressions, default and static methods in interfaces, improved collection, and JavaScript runtime. Also integrated JavaFX graphics subsystem.

Java SE 9 (JDK 9) (September 21, 2017): Introduced modularization of the JDK (module) under project Jigsaw, the Java Shell (jshell), and more.

Java SE 10 (18.3) (JDK 10) (March 2018): Introduced var for type inference local variable (similar to JavaScript). Introduced time-based release versioning with 2 releases each year, in March and September, denoted as YY.M. Removed native-header generation tool javah.

Java SE 11 (18.9) (LTS) (JDK 11) (September 2018): Extended var to lambda expression. Standardize HTTP client in java.net.http. Support TLS 1.3. Clean up the JDK and the installation package (removed JavaFX, JavaEE, CORBA modules, deprecated Nashorn JavaScript engine).

Java SE 12 (19.3) (JDK 12) (March 2019)

Installing JDK

In order to install the JDK on Windows, the following steps are followed:

Remove/uninstall the older version of JDK

It is highly recommended that we should install just the newest JDK. Even though we can install several versions of JDKat the same time but it is untidy and may unnecessarily confuse us while running the Java programs.

The earlier installed version of JDK can be, uninstalled by clicking on Control Panel | Programs|Programs and Features. Click on uninstall ALL programs begin with Java, such as Java SE Development Kit…, Java SE Runtime…, Java X Update…, and many more.

Step 2: Download the latest version of JDK

GotoJava SE download site:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Under Java Platform, Standard Edition|Java SE 11.0 and click the Oracle JDK |Download button.

Under Java SE Development Kit 11.0 and check Accept License Agreement.

Choose the JDK for your operating system, that is, Windows with 64 bit or 32 bit. Download the exe installer (for example,jdk-11.0.{x}_windows-x64_bin.exe about 150MB).

Step 3: Install the JDK

Run the downloaded installer (for example, jdk-11.0._windows-x64_bin.exe), which installs both the JDK and JRE.

By default, JDK is installed in directory C:\Program Files\Java\jdk-11.0.{x}, agree to the defaults and pursue the screen directions to install the latest version of JDK.

Use your File Explorer, navigate to C:\Program Files\Java to inspect the

subdirectories. Take note of your JDK installed directory, in particular, the update number {x}, which you will need in the next step.

In the following diagram, the JDK installed directory is C:\Program Files\Java\jdk1.8.0_131:

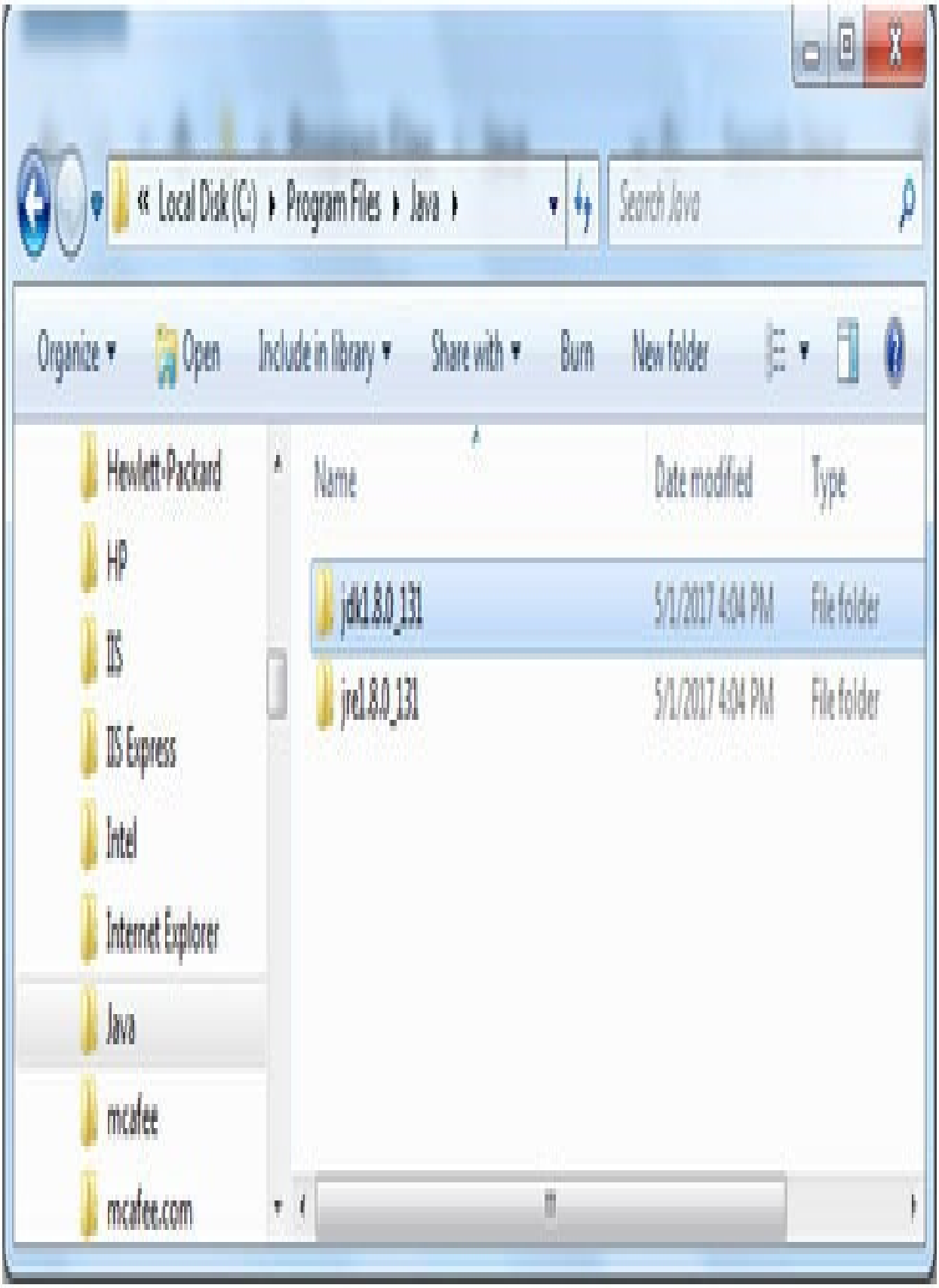

« Local Disk (C:) ▸ Program Files ▸ Java ▸
Search Java
Organize
Open
Include in library
Share with
Burn
New folder
Hewlett-Packard
HP
IIS
IIS Express
Intel
Internet Explorer
Java
mcafee
mcafee.com
Name
Date modified
Type
jdk1.8.0_131
5/1/2017 4:04 PM
File folder
jre1.8.0_131
5/1/2017 4:04 PM
File folder

Figure 1.1: The Java installed directories

Configuring Java development environment

Windows' Command Prompt (CMD) looks for the path environment variables or system variables for executing the various Java programs. The executable programs responsible for executing the java programssuch as Java compiler javac.exe and Java runtime java.exe reside in the subdirectory bin of the JDK installed directory. This bin folder of JDK must be in the PATH to run the Java programs.

To edit the PATH environment variable in Windows 10:

Launch Control Panel|System and Security|System. Click Advanced system settings on the left pane.

Switch to Advanced tab and click Environment Variables button.

Under System Variables (the bottom pane), scroll down to select variable Path and click Edit….

For newer Windows 10:You shall see a TABLE listing all the existing PATH entries (if not, goto next step). Click New and click Browse and navigate to your JDK's bin directory, that is, c:\Program Files\Java\jdk-11.0.{x}\bin, where {x} is your installation update number. Select Move Up to move this entry all the way to the TOP. Skip the next step:

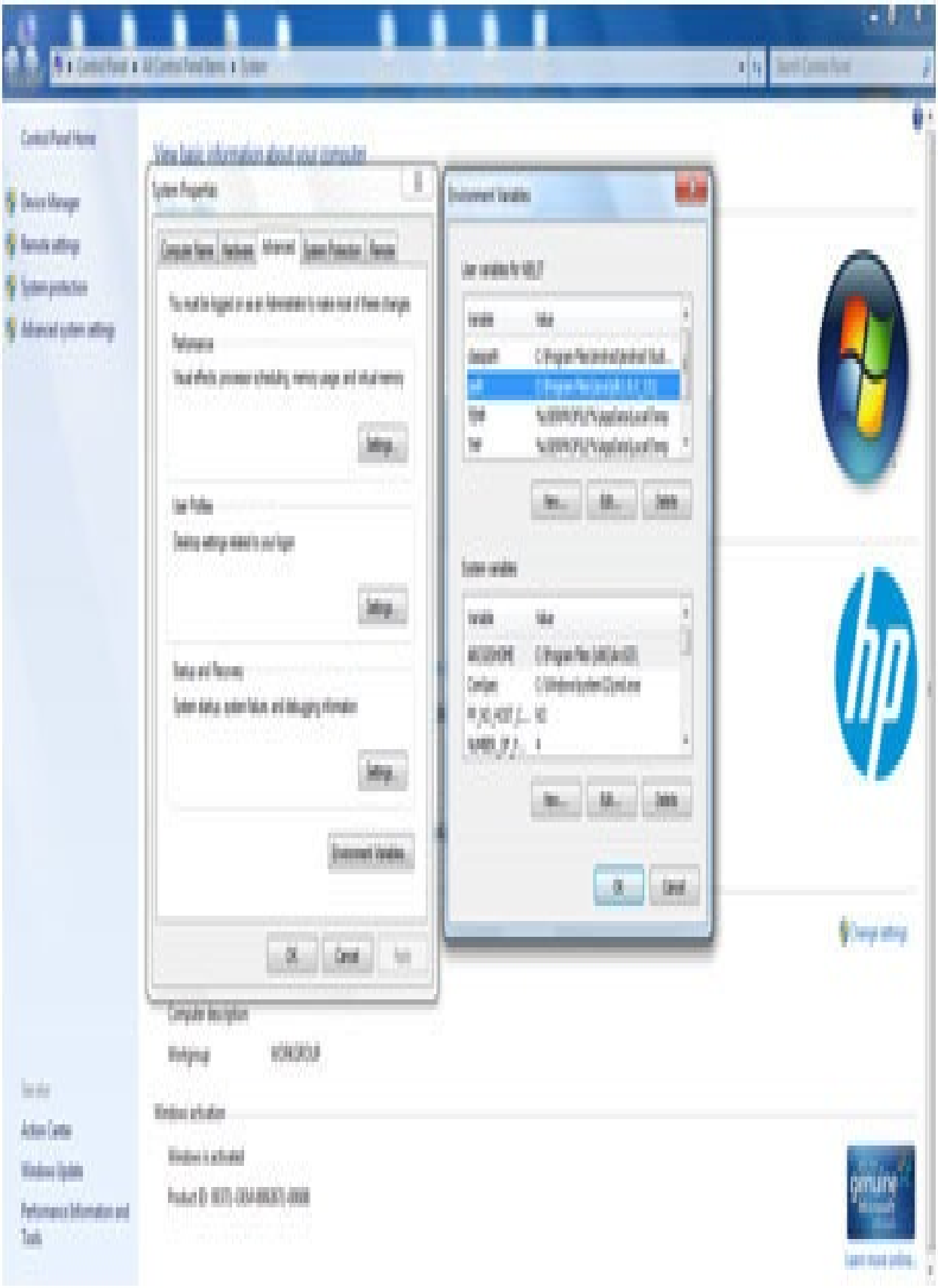


Figure 1.2: Setting the path

Verify the JDK installation

Launch a CMD via one of the following means:

Click Search button. Enter cmd and choose Command Prompt.

Right-click Start button and enter run enter cmd, or

Click Start button. Search Windows System|Command Prompt.

Issue the following commands to verify your JDK installation:

Issue path command to list the contents of the PATH environment variable. Check to make sure that your JDK's bin is listed in the PATH:

path

PATH=c:\Program Files\Java\jdk-11.0.{x}\bin;[other entries…]

Issue the following commands to verify that JDK/JRE are properly installed and display their version:

// Display the JDK version

javac -version

javac 11.0.1

// Display the JRE version

java -version

java version “11.0.1” 2018-10-16 LTS

Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)

Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)

The Eclipse IDE

To write our Java programs, we will need a text editor. On Windows machine, you can use any simple text editor like Notepad or TextPad to write the Java programs. The compilation and execution of programs can be carried out through the command prompt to be discussed in the next section. There are even more sophisticated IDEs available in the market like the NetBeans editor. A Java IDE that is open-source and free, which can be downloaded from http://www.netbeans.org/index.html. However, for now, we can consider using the Eclipse IDE. All the programs in this book are written and executed on the Eclipse IDE. For writing the Java applications, the preferred IDE is Eclipse, amulti-language software developmentenvironment featuring an extensible plug-in system. It can be used to develop various types of applications, using languages such as Java, Ada, C, C++, COBOL, Python, and more. For Java development, we can download the Eclipse IDE for Java EE Developers (www.eclipse.org/downloads/packages/eclipse-ide-java-eedevelopers/heliossr1). Several editions are available: Windows (32 and 64-bit),Mac OS X (Cocoa 32 and 64), and Linux (32 and 64-bit). Simply select the relevantone for your operating system. All the examples in this book were testedusing the version of Eclipse for Windows.


About Eclipse
Eclipse IDE for Java Developers
Version: Neon.3 Release (4.6.3)
Build id: 20170314-1500
(c) Copyright Eclipse contributors and others 2000, 2017. All rights reserved. Eclipse and the Eclipse logo are trademarks of the Eclipse Foundation, Inc., https://www.eclipse.org/. The Eclipse logo cannot be altered without Eclipse's permission. Eclipse logos are provided for use under the Eclipse logo and trademark guidelines, https://www.eclipse.org/logotm/. Oracle and Java are trademarks or registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.
This product includes software developed by other open source projects including the Apache Software Foundation, https://www.apache.org/.
Installation Details
OK

Figure 1.3: About Eclipse

Eclipse adopts the idea of a workspace. A workspace is a folder that you have elected to accumulate all your projects. When we first start Eclipse, we are impelled to select a workspace as shown in Figure 1.4:

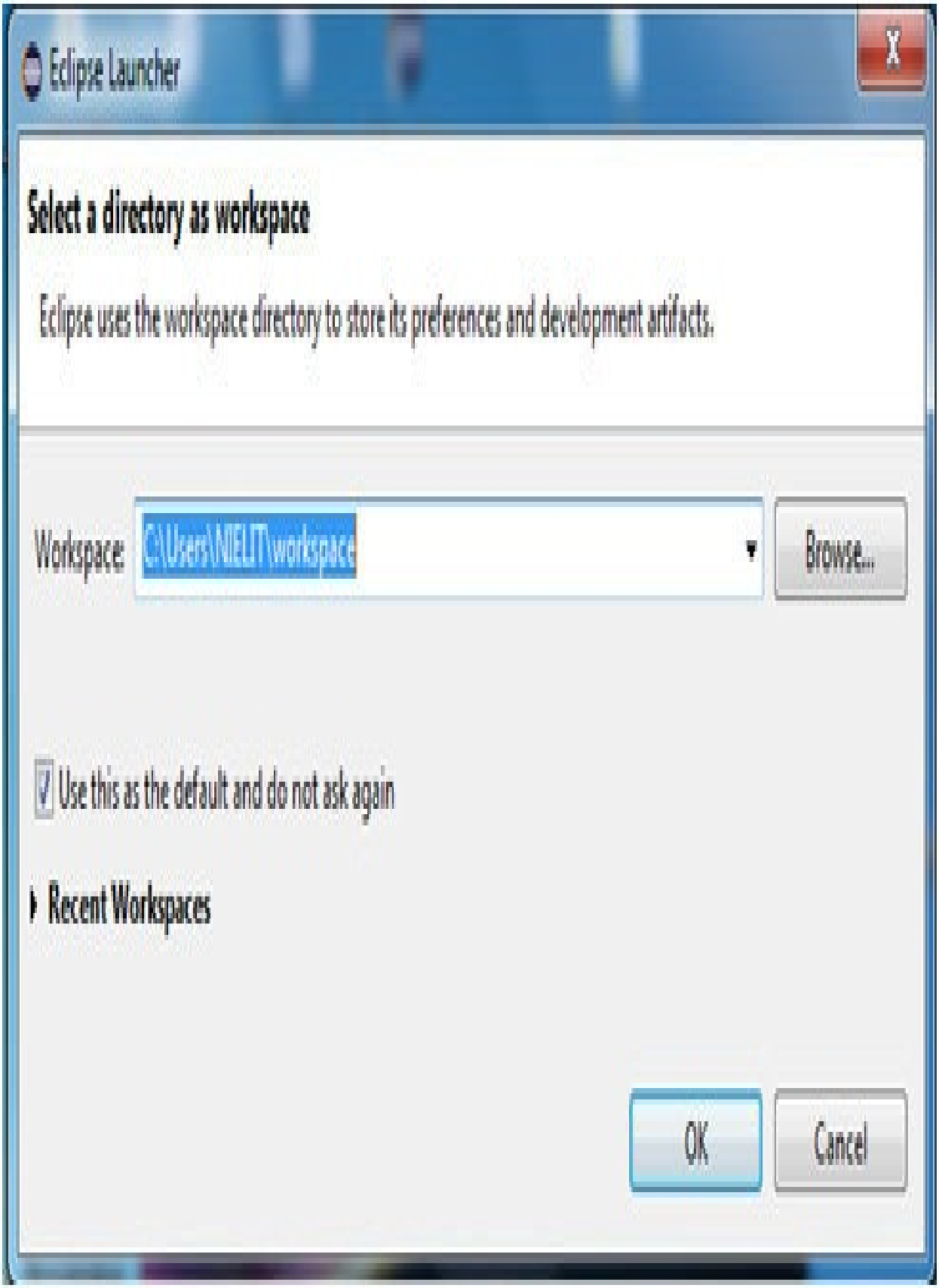

Eclipse Launcher
Select a directory as workspace
Eclipse uses the workspace directory to store its preferences and development artifacts.
Workspace:
C:\Users\NIELIT\workspace
Browse...
Use this as the default and do not ask again
Recent Workspaces
OK
Cancel

Figure 1.4: Eclipse launcher with workspace details

When Eclipse has completed launching the projects situated in your workspace, it will display a number of panes in the IDE as shown in Figure 1.5:


package book;
import java.io.File;
class FileDemo {
static void p(String s) {
System.out.println(s);
}
public static void main(String args[]) {
File f1 = new File("C:/Users/NIELIT/workspace/book/dem
p("Name of the File " + f1.getName());
p("Path of the File " + f1.getPath());
p("Abs Path of the File: " + f1.getAbsolutePath());
p("Parent directory " + f1.getParent());
p(f1.exists() ? "Available" : "Not Available");
p(f1.canWrite() ? "is writeable" : "is not writeable")
p(f1.canRead() ? "is readable" : "is not readable");
p("is " + (f1.isDirectory() ? "" : "not" + " a directo
p(f1.isFile() ? "is a normal file" : "might be a named
p(f1.isAbsolute() ? "is absolute" : "is not absolute")
p("File last modified is: " + f1.lastModified());
p("File size in Bytes: " + f1.length() + " Bytes");
}
Name of the File d
Path of the File C
Abs Path of the Fi
Parent directory C
Not Available
is not writeable
is not readable
is not a directory
might be a named p
is absolute
File last modified
File size in Bytes

Figure 1.5: Eclipse IDE with different panes

The subsequent sections emphasize on some of the significant panes that you require to be familiar with while developing Java applications.

Package Explorer

The Package Explorer, shown in Figure 1.6, lists all the projects presently in your workspace:

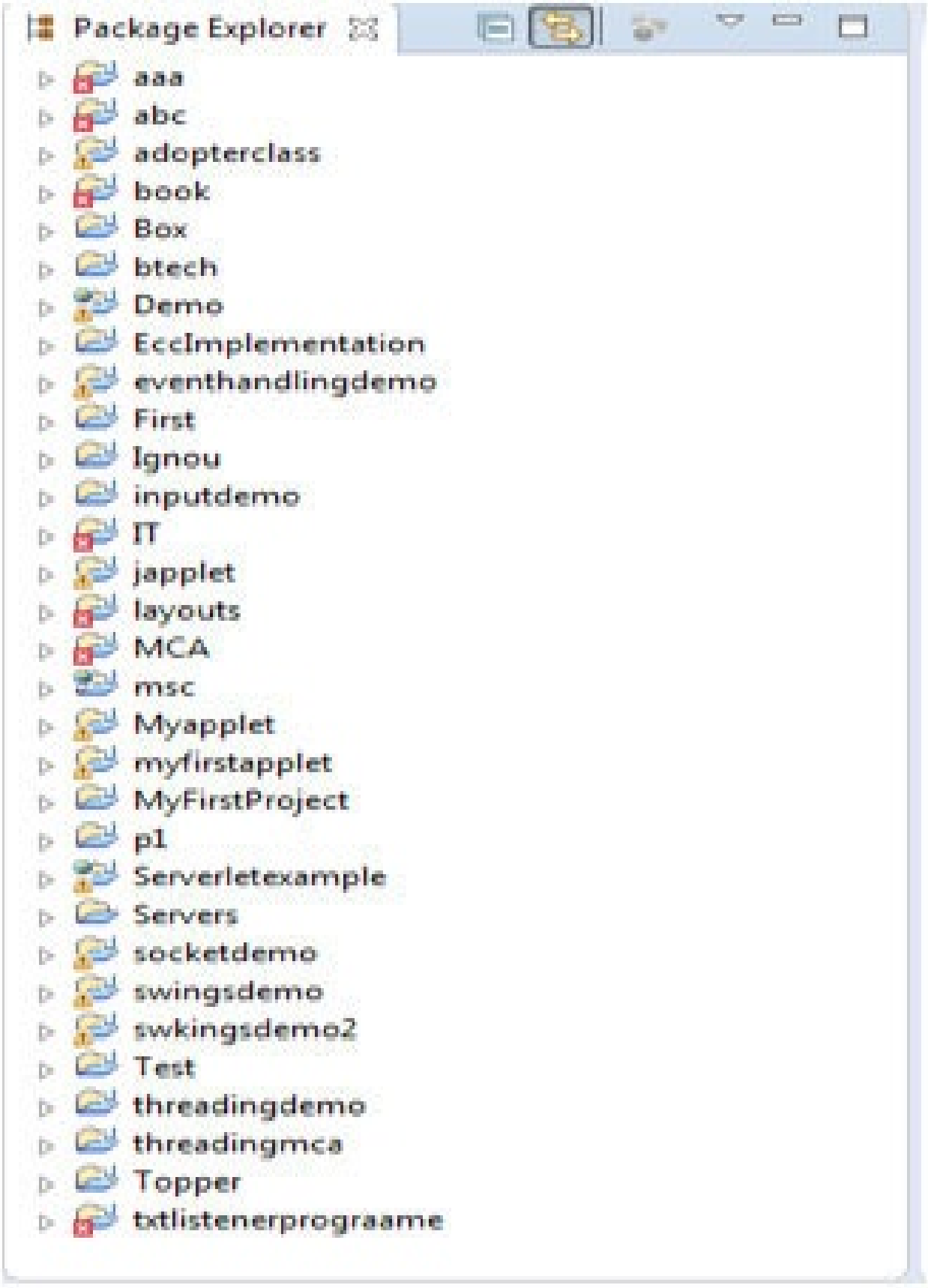

Package Explorer
aaa
abc
adopterclass
book
Box
btech
Demo
EccImplementation
eventhandlingdemo
First
Ignou
inputdemo
IT
japplet
layouts
MCA
msc
Myapplet
myfirstapplet
MyFirstProject
p1
Serverletexample
Servers
socketdemo
swingsdemo
swkingsdemo2
Test
threadingdemo
threadingmca
Topper
txtlistenerprograame

Figure 1.6: A package explorer

To amend a particular item in your project, you can double-click on it and the file will be displayed in the relevant editor. You can also right-click on every item displayed in the Package Explorer to display context sensitive menu(s) associated to the chosen item. For example, if you wish to add a new .java file to the project, you can right click on the package name in the Package Explorer and then select New| Class as shown in Figure 1.7:

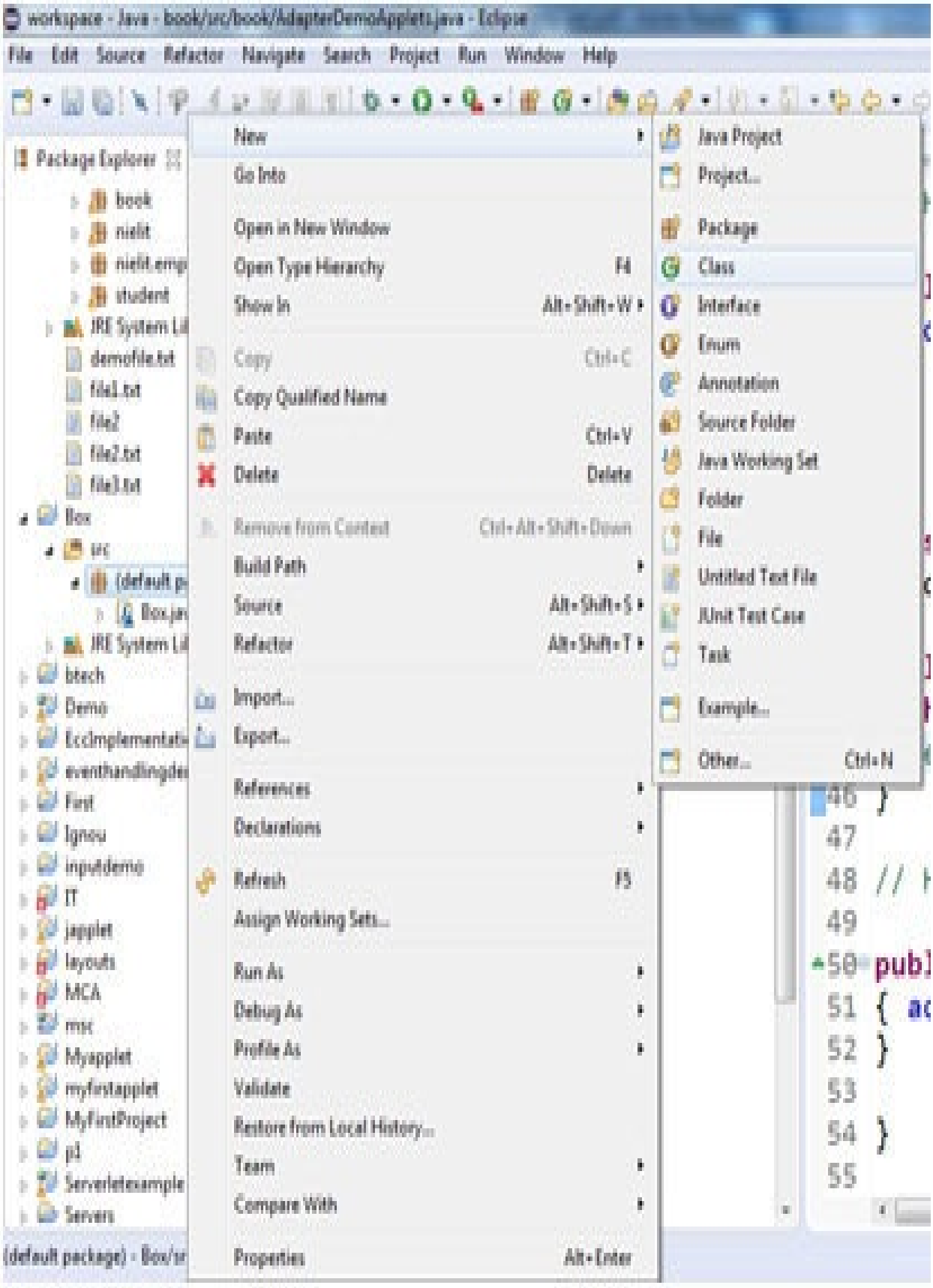

workspace - Java - book/src/book/AdapterDemoApplets.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
book
nielit
student
demofile.txt
file1.txt
file2
file2.txt
file3.txt
Box
src
Box.jav
btech
Demo
First
Ignou
inputdemo
IT
japplet
layouts
MCA
msc
Myapplet
myfirstapplet
MyFirstProject
pl
Servletexample
Servers
(default package) - Box/sr
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
References
Declarations
Refresh F5
Assign Working Sets...
Run As
Debug As
Profile As
Validate
Restore from Local History...
Team
Compare With
Properties Alt+Enter
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N
46 }
47
48 //
49
50 pub
51 { a
52 }
53
54 }
55

Figure 1.7: Creating a new class

Java projects from other workspaces

There may perhaps be times when you encompassnumerous workspaces formed to stock updiverse projects. If you require to access the project in a different workspace, there are commonly two ways to go about doing so. First, you can change to the preferred workspace by selecting File|Switch Workspace as in Figure 1.8. Identify the new workspace to work on and then restart Eclipse:

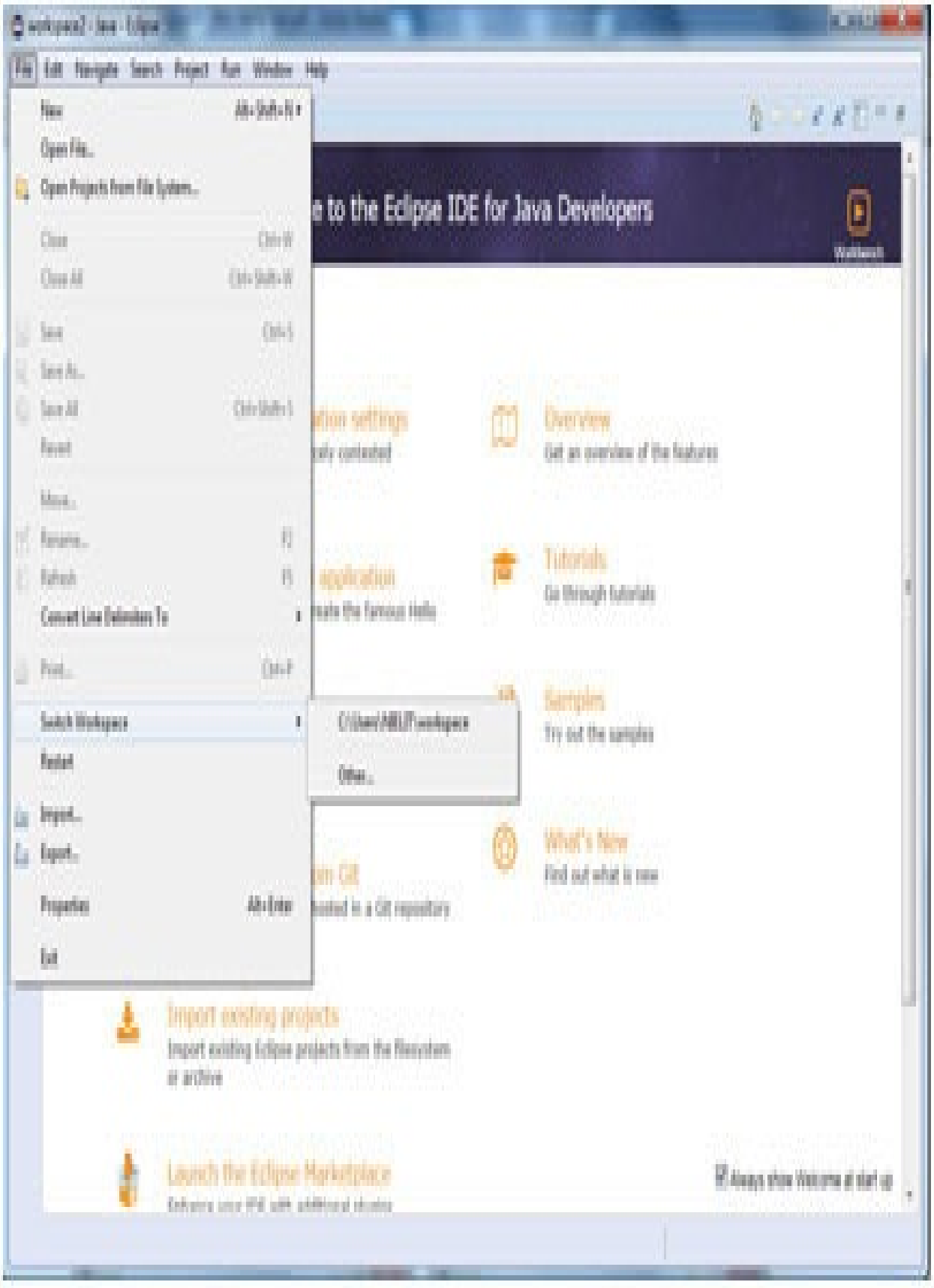

File Edit Navigate Search Project Run Window Help
New
Open File...
Open Projects from File System...
Close
Close All
Save
Save As...
Save All
Revert
Move...
Rename...
Refresh
Convert Line Delimiters To
Print...
Switch Workspace
Restart
Import...
Export...
Properties
Exit
Other...
to the Eclipse IDE for Java Developers
Overview
Get an overview of the features
Tutorials
Go through tutorials
Samples
Try out the samples
What's New
Find out what is new
Import existing projects
Import existing Eclipse projects from the filesystem or archive
Launch the Eclipse Marketplace
Always show Welcome at start up

Figure 1.8: Switching the namespace

The second approach is to import the existing project by using the import option as in Figure 1.9:

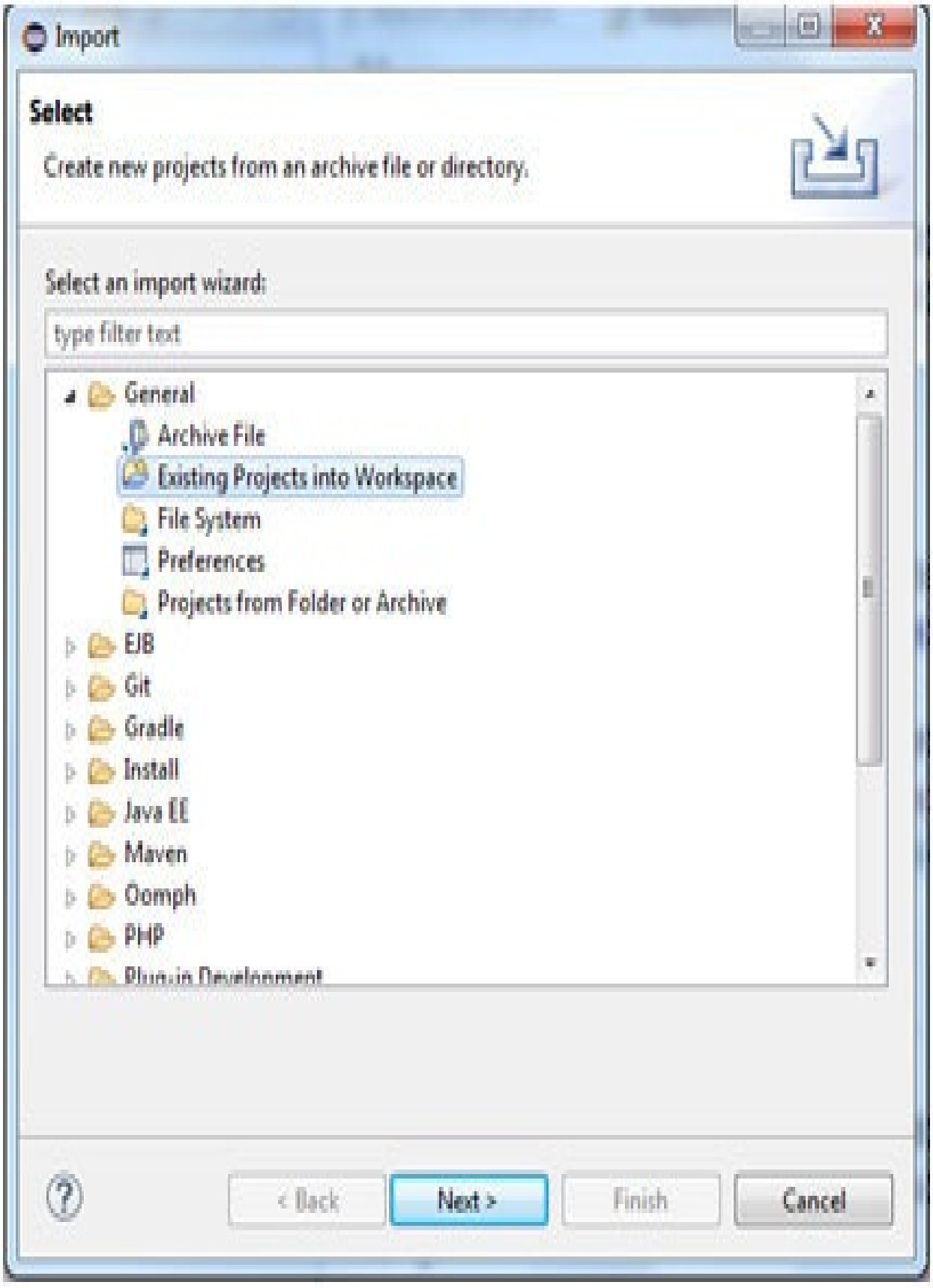
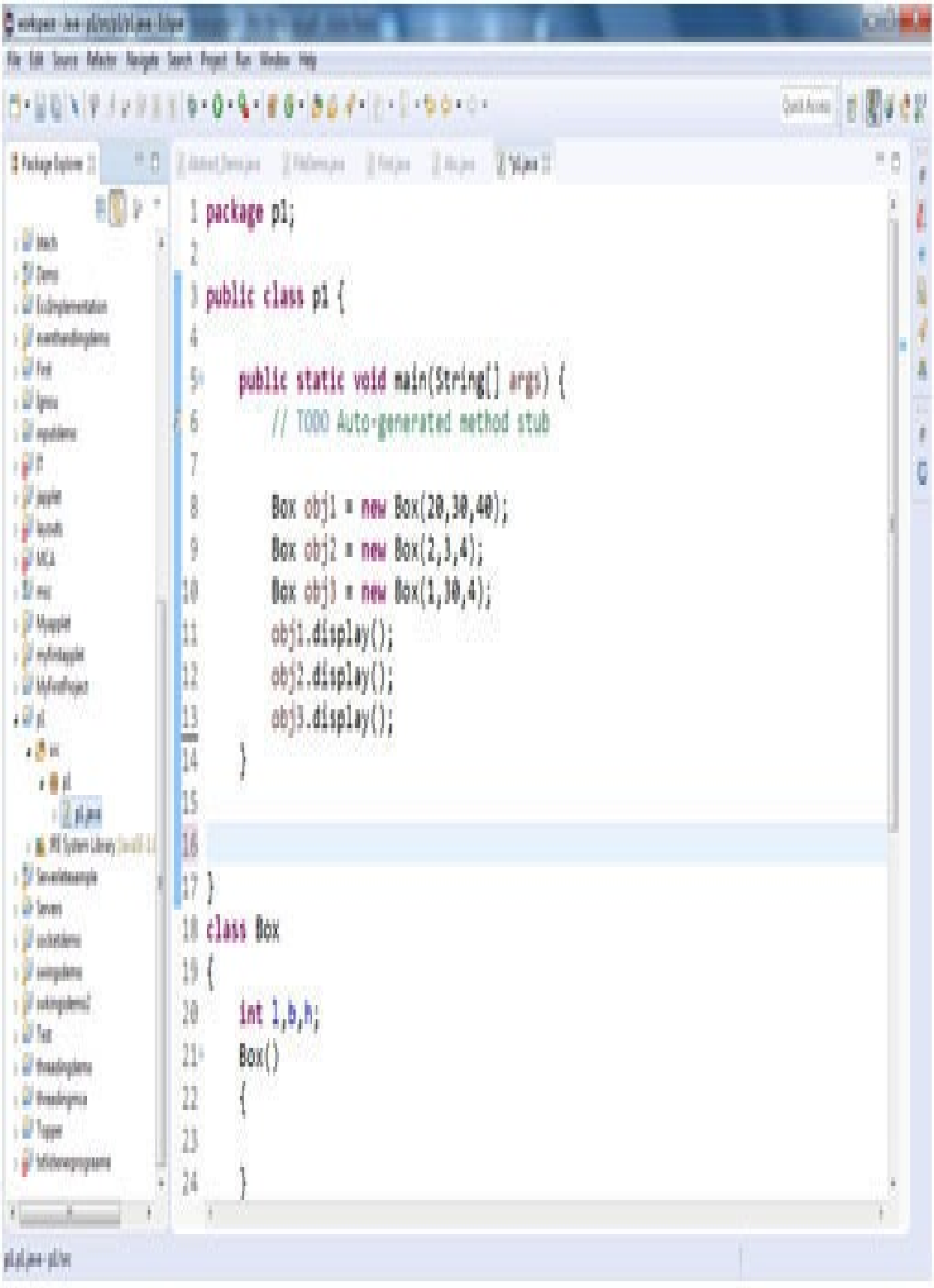

Import
Select
Create new projects from an archive file or directory.
Select an import wizard:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
Projects from Folder or Archive
EJB
Git
Gradle
Install
Java EE
Maven
Oomph
PHP
< Back
Next >
Finish
Cancel

Figure 1.9: Importing the existing project

Editors

Depending on the type of items you have double-clicked in the Package Explorer, Eclipse will open the suitable editor for you to edit the file. For example, if you double-click on a .java file, the text editor for editing the source file will be opened as shown in Figure 1.10:



```
package p1;

public class p1 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Box obj1 = new Box(20,30,40);
        Box obj2 = new Box(2,3,4);
        Box obj3 = new Box(1,30,4);
        obj1.display();
        obj2.display();
        obj3.display();
    }

}
class Box
{
    int l,b,h;
    Box()
    {

    }
```

Figure 1.10: Editor to write programs

Refactoring

Refactoring is a very helpfulcharacteristic that most recent IDEs support. Eclipse supports a complete slew of refactoring features that construct application development well organized and efficiently. In Eclipse, when you place the cursor at a particularobject/variable, the editor will draw attention to all occurrences of the chosen object in the current source code as shown in Figure 1.11:

```
Box obj1 = new Box(20,30,40);
Box obj2 = new Box(2,3,4);
Box obj3 = new Box(1,30,4);
obj1.display();
obj2.display();
obj3.display();
```

Figure 1.11: Refactoring in Eclipse IDE

This quality is very helpful for identifying wherever a particular object is used in your code. To modify the name of an object, simply right-click on it and select Refactor| Rename… as shown in Figure 1.12.

After entering a new name for the object, all occurrences of the object will be changed dynamically:

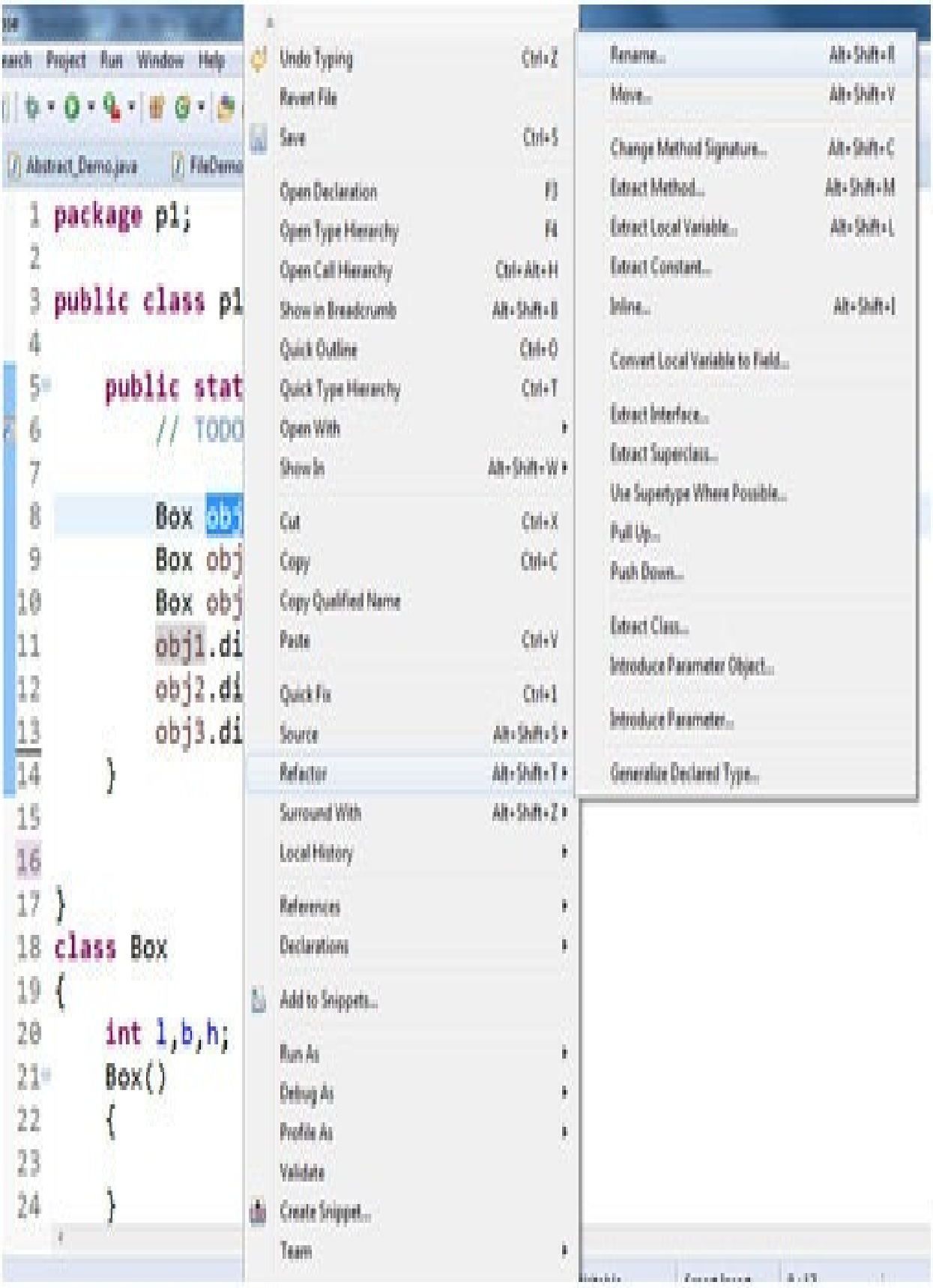

search Project Run Window Help
Abstract_Demo.java
FileDemo
1 package p1;
2
3 public class p1
4
5 public stat
6 // TODO
7
8 Box obj
9 Box obj
10 Box obj
11 obj1.di
12 obj2.di
13 obj3.di
14 }
15
16
17 }
18 class Box
19 {
20 int l,b,h;
21 Box()
22 {
23
24 }
Undo Typing Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show in Breadcrumb Alt+Shift+B
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Surround With Alt+Shift+Z
Local History
References
Declarations
Add to Snippets...
Run As
Debug As
Profile As
Validate
Create Snippet...
Team
Rename... Alt+Shift+R
Move... Alt+Shift+V
Change Method Signature... Alt+Shift+C
Extract Method... Alt+Shift+M
Extract Local Variable... Alt+Shift+L
Extract Constant...
Inline... Alt+Shift+I
Convert Local Variable to Field...
Extract Interface...
Extract Superclass...
Use Supertype Where Possible...
Pull Up...
Push Down...
Extract Class...
Introduce Parameter Object...
Introduce Parameter...
Generalize Declared Type...

Figure 1.12: Renaming all the occurrences of an object

Debugging

In order to debug a Java application, press F11 in Eclipse IDE. The program will generate the output if there are no errors and program compiles normally.

If we want to properly inspect our Java program we can set breakpoints-way to momentarily pause the finishing of the application and after that verify the values of variables and objects in the Java code window. A breakpoint can be set by double-clicking on the left-most column in the code editor. of eclipse IDE. Figure 1.13 shows a breakpoint set on a particular statement.

While the application is running and the initial breakpoint is reached, Eclipse will display a Confirm Perspective Switch dialog. Essentially, it requests to switch to the Debug perspective. To avert this window from appearing yet again, check the Remember my decision checkbox at the bottom and click Yes.

Eclipse now highlights the breakpoint. At this point, you can right-click on any selected object or a variable and view its content using the various options like Watch, Inspect, and Display as shown in Figure 1.13:

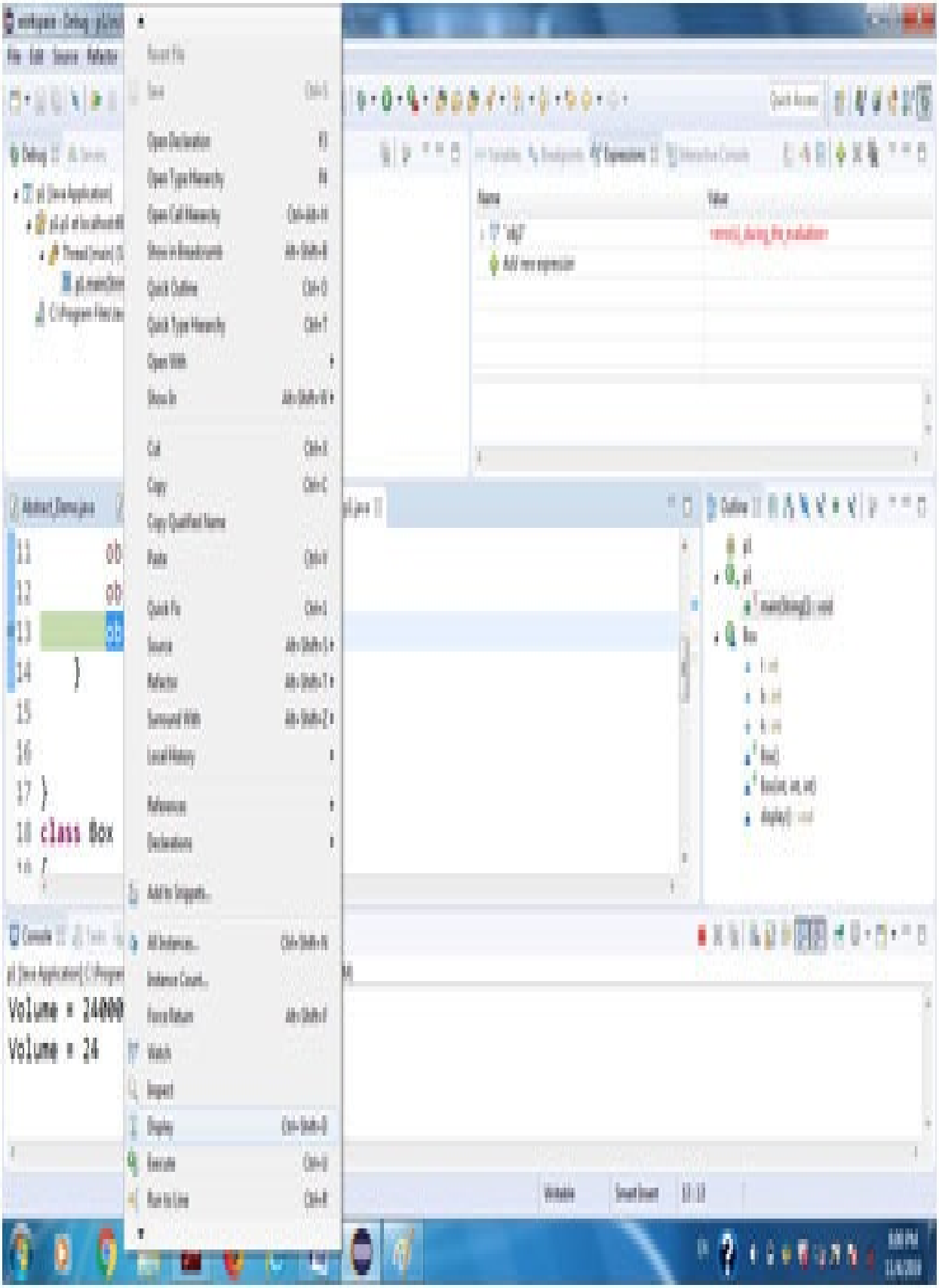

Volume = 24000
Volume = 24

Figure 1.13: Setting the breakpoint and viewing the contents

There are a number of options at this point to carry on the execution:

Step into: Press F5 to step into the next method call/statement.

Step over: Press F6 to step over the next method call without entering it.

Step return: Press F7 to return from a method that has been stepped into.

Resume execution: Press F8 to resume the execution.

Assignments

How is Java more secure than C and C++?

What is a bytecode file in Java and how does it make a difference?

How to set the PATH to execute the Java programs?

Some features of C and C++ are removed from Java. List those features and explain how it positively affects the Java language?

How to debug a Java program?

What is a breakpoint? Create a breakpoint to check the value of a variable?

CHAPTER 2

Basics of Java Programming

Introduction

As in most of the programming languages, the components in Java are not working in isolation, rather they are interdependent to make a meaningful language. The understanding of Java language depends on several concepts mostly object oriented concepts. The Java has its syntax from C and object oriented features from C++.

So for a programmer with knowledge of C and C++ has a better chance to understand the Java and develop some good application using this platform. This chapter begins with the structure of Java program to explain the programmers the different parts of a Java program. Some of the sections are optional and can be used depending upon the need of the programmer. In this chapter, the basic program is compiled and the working of JDK is explained. The basic constructs like data types and keywords are explained in this unit. The control structures are the basic building blocks of any of the programming language; these structures are explained in detail to help the beginners to write some basic programs.

Structure

Introduction

Structure of Java program

A simple Java program

Executing the program

Data types, variables, and keywords in Java

Control structures in Java

Command line arguments

Creating basic programmes in Java

Assignments

Objective

This chapter explains the basic structure of Java program. After reading this chapter, a reader should be able to write some basic programs and should handle the command line arguments.

Structure of Java program

Java programs have different sections to deal with different functionalities. Some of the sections are mandatory like the class, which includes the main method, and most of the sections are optional like importing a package as it depends on the need of the programmer to use any of the package. The different sections of Java program shall be explained in detail in other chapters and this chapter only gives the overall structure of Java program.

The different sections of Java program are as under:

Documentation section: This section is optional and may include the detailed description of the program in comments. As in C language the comments can be a single line with // or a block with /* */. For example the below lines describe the details about any of the Java program:

/* This Java program gets a number from a command line and checks whether the number is a prime number or not

*/

Package section: This section is optional and the only purpose of this section is to include your class with any of the packages. It isup to the programmer to include the name of the package or to leave it. The syntax for writing the package name is:

```
package nielit;
```

Therefore, the program class that we are going to create will be within the package nielit.

Import section: This section is optional and can be used to import any of the existing packages, which are inbuilt, or user defined. The syntax for importing any of the package is :

```
import java.io.*; //imports all classes in input output package

import java.awt.Button; //imports only Button class from awt package

import nielit.*; //imports all the classes of nielit package created by the programmer
```

Interface section: This section is optional and includes the interfaces, which are used for implementation of multiple inheritances. The interfaces include the abstract classes and the final data (To be explained in Chapter 4: Packages and Interfaces). The example of interface is:

```
interface MyInterface

{

void display();

void show();
```

```
final int data=100;

}
```

Class definitions section: As our Java program may consist of multiple classes, this section includes the definition of those classes. This section is also optional as we may have only a single class with main method. For example:

```
//This class is optional

class MyClassA

{

// Data + Methods

}

// This class is mandatory as it includes the main method

class MyClassB

{

// Data + Methods

// main method

public static void main( String arg[])

{

}

}
```

Class with main method: As the execution of every Java program starts with the main method, this class is mandatory as it includes the main method where from the execution starts.

```
class MainClassB

{

// Data + Methods

// main method

public static void main( String arg[])

{

}

}
```

The following is the basic program with different sections:

```
  /*            Documentation Section   -------Optional      */
package nielit;                          // Optional
import java.awt.*;                      //Optional
interface myint                          //Optional
{
}
class  Mca                               //Optional
{
}
class MainClass
{
public static void main(String arg[])
  {
  }
}
```

The only mandatory section is the class where the main method is incorporated. The detailed description of these sections will be given in the other chapters.

A first simple Java program

We should start writing the programs in order to understand the different components of Java language:

```
/* This is our first Java program that explains the various elements of Java */

class MyExample {

// This is the main class which includes the main method

public static void main (String arg[])

{

System.out.println (" This is my first Java program");

}// closing brace for main method

}//closing brace for class
```

The various components of this simple program are explained as under:

class MyExample: As Java is a true object oriented language, it is mandatory to have at least one class, which will have a main method. The MyExample class in our example is a user defined class and must follow all the conventions of naming an identifier as in C. As Java is a case sensitive language like C, it is pertinent to mention here that the class names start with upper case and all words within that class name also start with the upper case letters. Some of the examples of built in classes in Javaare: String, Exception, ArrayIndexOutOfException and many more. This is the reason that the only

upper case words in the above program are MyExample, String, and System as all of these are classes.

public: The different access specifiers like public, private, and protected are used with the data members and the methods in Java. The main method must be accessed by the run time environment of Java so it must be declared as public. As the default visibility in Java is friendly and not private as in C++. The default access specifier restricts the access to a main method and as such it must be declared with public so that it is accessible to everyone. The access specifiers are explained with more detail in upcoming chapters.

static: In object-oriented programming (OOP), we create objects of a class to access its methods but what if we want to access a method without creating objects. For those situations where we can use the class names independently without an object, we use static members. The most important example is the main() method which is declared as static, the reason being that it must be called before any object is created. The static can be used with the methods or the data members. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. These static data members are shared by all the objects and are accessed by the class names directly. The below example calls a parseInt method of Integer class without creating the object:

```
int i = Integer.parseInt("20");
```

void main:As in any other programming language, the void is the return type and the main is the keyword used to specify the main method.

System.out.println(""): This statement prints the message in double quotes. The System class is used as everything in Java is executed through classes and

objects. The println method displays the message and takes the cursor to the next line.

Executing the program

In order to understand the basic idea of executing the Java program, we will try to find out the procedure to execute the programmes through Command Prompt using Notepad. To execute the program we need two important files javac (Compiler) and java (Interpreter). The steps for running a program in notepad are as under:

Open Notepad.

Write a the following programme in Notepad:

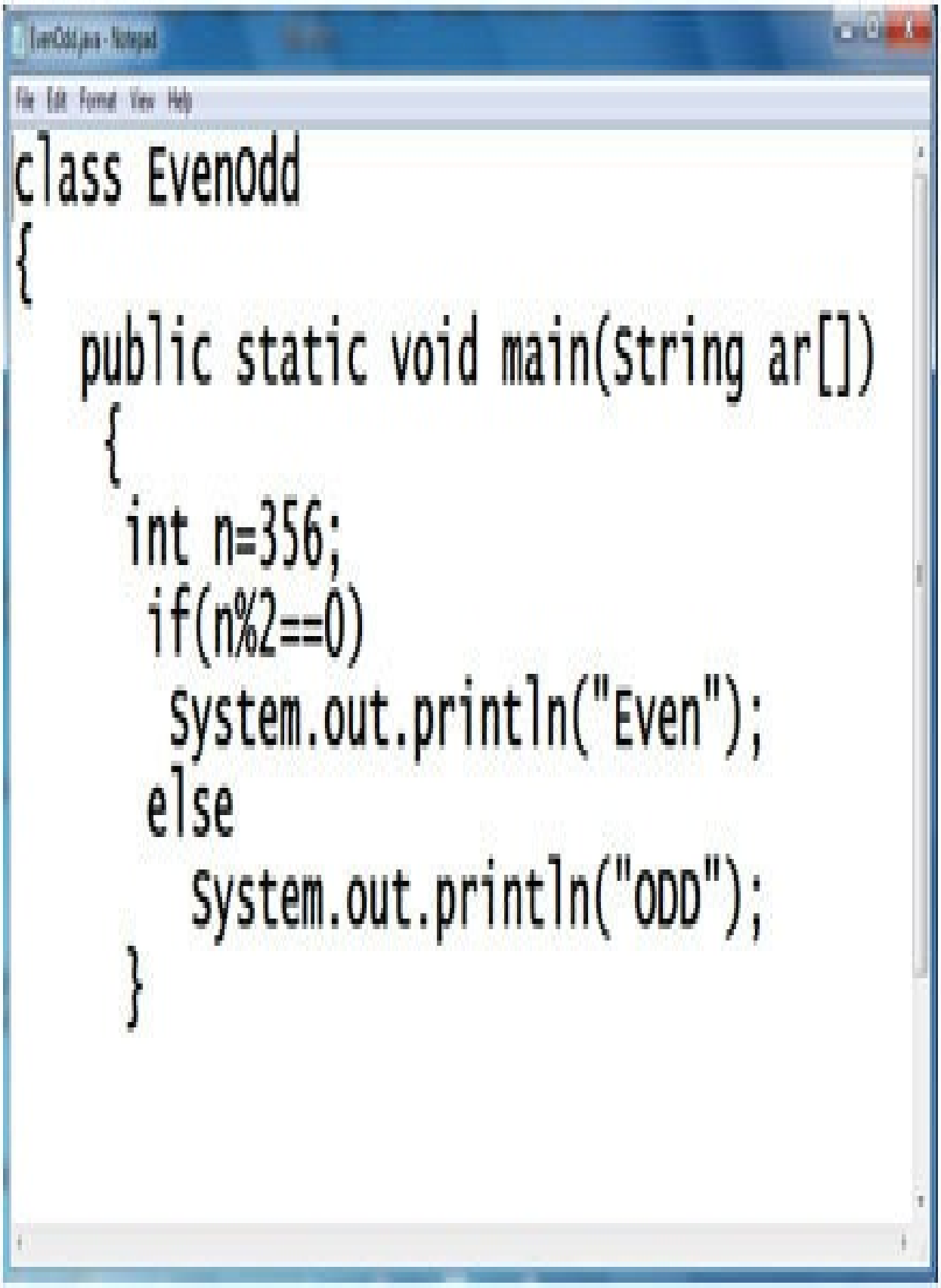

File Edit Format View Help
class Evenodd
{
public static void main(String ar[])
{
int n=356;
if(n%2==0)
System.out.println("Even");
else
System.out.println("ODD");
}

Figure 2.1: A Notepad with basic Java Program

If the path is not set, save the programme at the path where Java is installed as in our case the path is:

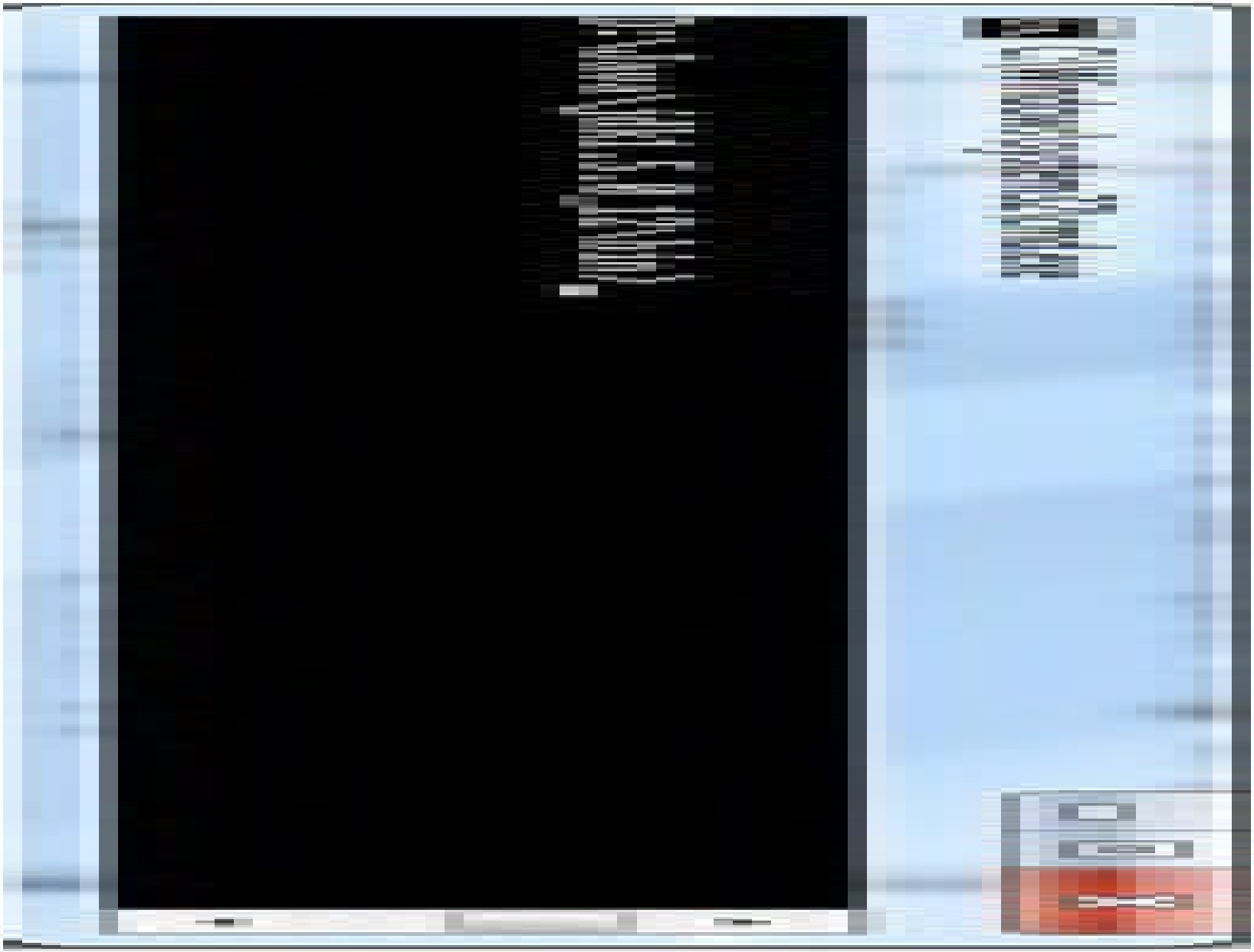


Figure 2.2

Now compile the programme using the command. If there are no errors, the bytecode file with .class extension will be generated:

javac EvenOdd.java

After compiling the program successfully, the following command is used to generate the output:

java EvenOdd

Key points

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

Casesensitivity: Java is case sensitive, which means identifierJava and java 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 word's first letter should be in upper Case. For 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 word's first letter should be in upper case. For 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 and append .java to the end of the name. For 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.

Data types in Java

As in other programming languages, Java has primitive and non-primitive data types. The primitive types can be modified with basic computer instructions. The eight primitive types of data available in Java are byte, short, int, long, char, float, double, and boolean. These can be put in four major categories:

Integers: This group includes byte, short, int, and long, which are for whole-valued signed numbers. For example:

```
int x,y,z;

byte s1,s2;
```

Floating-point numbers: This group includes float and double, which represent numbers with fractional precision. For example:

```
float f1;

double var1;
```

Characters: This group includes char, which represents symbols in a character set, like letters and numbers. For example:

char ch='A';

Boolean: This group includes boolean, which is a unique type for representing true/false values. For example:

boolean b;

The primitive types are used to create the data members in the class and final data in the interface. These data members can also be used to create the arrays of any of the types for example the following statement creates the array of int data type:

int a[] = new int[10];

The primitive types are definite to have aclear range and numerical behavior. Java is different from C and C++ because of Java's portability requirement; all data types have a firmlydefinite range. For example, an int is for all times 32 bits, in spite of the particular platform. This allows programs to be written that are guaranteed to run exclusive of porting on any device architecture. While firmly specifying the size of an integer could cause a little loss of performance in some environments, it is essential in order to get portability of Java programs.

In addition to the primitive data types, Java also has the type wrappers to give additional flexibility to the programmers to deal with different programming situations. The type wrappers are the classes for the corresponding data types like Integer for int, Float for float and so on. These classes have a number of methods to help programmer to execute multiple functionalities. The range of these primitive data types in terms of number of bits is as under:

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Table 2.1: Range of primitive types

Name	Width in bits	Approximate range
Double	64	4.9e–324 to 1.8e+308
Float	32	1.4e–045 to 3.4e+038

Table 2.2: Range of primitive types

Variables in Java

As in other programming languages, Java makes use of a variable, which is the fundamental element of storage in a Java program. A variable is created by the mixture of an identifier, a type, and an initialization, which is optional. In addition, all variables have a scope, which defines their visibility, and a lifetime. The basic variable is created with the following syntax:

type identifier1 [= value, identifier2 [= value] …] ;

int var1=10,var2=100;

// Creation of var1 and var2 variables which are initialized with 10 and 100 respectively

We can create nnumber of variables separated with comma. The variables may be initialized as in above case var1 is initialized with 10 and var2 with 100. Here are some more examples of variable declaration:

float f1; // declares a float variable

int var1=30, var2, var3=100; // declares three variables and initializing only var1 and var2

char ch= 'Y'; // declares ch as char and initialize it with Y

byte b; // declare a byte variable

double d1=3.14; // declare a double variable and initialize it

Although the preceding examples have used only constants as initializers, Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared. For example, here is a short program that calculates the value of z dynamically:

```
class DynInitDemo {

public static void main(String args[])

{

double x = 6.0, y = 8.0;

double z = Math.sqrt(x * x + y * y);

System.out.println("Hypotenuse= " + z);

}

}
```

Keywords in Java

The keyword is a reserved word that cannot be used as a name of a variable, class name, interface name, package name, or method name as in any other language like C and C++. These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language. The following keywords are used in Java:

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
Boolean	do	if	private	this
break	double	implements	protected	Throw
byte	else	import	public	Throws
case	enum	instanceof	return	Transient
catch	extends	int	short	Try
char	final	inter face	static	Void
class	finally	long	strictfp	Volatile
const	float	native	super	While

Table 2.3: Keywords in Java

The following program makes use of data types, variables, and keywords defined in the above sections:

```
package book;

public class DemoVariables {

public static void main(String[] args) {

int var_a[] = { 5,7,10}; // An array of int data type with three items

double d = 12.8; // double type variable d

double c;

System.out.println("The addition of int array and double variable");

for (int i=0;i<var_a.length;i++)

{

c= var_a[i]+ d;

System.out.println(var_a[i] +" + " + d + " = " + c);

}

}

}
```

The output for the above code is as follows:

The addition of int array and double variable

5 + 12.8 = 17.8

7 + 12.8 = 19.8

10 + 12.8 = 22.8

Control structures in Java

Java has never tried to reinvent the existing methodologies to do decision making or looping. As the syntax of Java is almost as that of C, it has all the control structures with the syntax as in C and C++. In this section we will discuss in detail the decision-making and looping structures with the help of a relevant Java program.

The if statement

The if statement, in order to control the flow of execution of various statements in a Java program, the if statement is considered as a powerful statement. The different forms of if statements are as under:

```
// if with one statement

if (condition )

Statement ;


// if with more than one statement

if (condition )

{

Statement 1 ;

Statement 2 ;

Statement N ;

}


// if with one statement and else

if (condition )

Statement 1 ;
```

```
else
Statement 2 ;

// if with more than one statement and else
if (condition )
{
Statement 1 ;
Statement 2 ;
Statement N ;
}
else
{
Statement 1 ;
Statement 2 ;
Statement N ;
}
```

The above statements can be understood by the below Java program:

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

```
int x, y;

x = 100;

y = 200;

if(x < y) System.out.println("Y is greater than X");

x = x * 2;

if(x == y) System.out.println("X now equal to Y");

x = x * 2;

if(x > y) System.out.println("X now greater than Y");

// this won't display anything

if(x == y) System.out.println("you won't see this");

}

}
```

The output for the above code is as follows:

Y is greater than X

X now equal to Y

X now greater than Y

The following example also uses the if statement:

```
public class IfElse
{
public static void main(String ar[])
{
int n=30;
if(n>40)
System.out.println("You are Old");
else
System.out.println("You are Young");
}
} // End of the program
```

The output will be You are Young.

We can implement the nested if…else statements as under:

```
if (Marks >75) && (sport_marks > 50)
Statement 1;
else
Statement 2;
```

The above if statement can be also written as:

```
if (Marks> 75)
{
if (sports_ marks> 50)
Statement 1;
}
else
Statement 2;
```

The switch statement

The if statement makes the things complicated when we have to choose from many alternatives. To have more simplicity in selecting from several choices based on the condition we can make use of the switch statement. The general structure of switch statement is as under:

```
switch (Expression)

{

case value1:

Statement 1;

Statement N:

break;

case value2:

Statement 1;

Statement N:

break;

case valueN:

Statement 1;

Statement N:

break;
```

```
default :

Statement 1;

Statement N:

break;

}
```

The below program illustrates the usage of switch statement. The user is given a choice and based on his choice appropriate case is executed:

```
import java.util.Scanner;

public class SwitchExample {

public static void main(String[] args) {

int a=30;

int b=20;

int c;

int choice;

while(true)

{

System.out.println(" Press 1 for Addition");

System.out.println(" Press 2 for Subtraction");

System.out.println(" Press 3 for Multiplication");
```

```
System.out.println(" Press 4 for Division");
System.out.println(" Press 5 for Exit");
Scanner s = new Scanner(System.in);
choice = Integer.parseInt(s.nextLine());
switch(choice)
{
case 1:
c=a+b;
System.out.println(" Addition =" + c);
break;
case 2:
c=a-b;
System.out.println(" Subtraction =" + c);
break;
case 3:
c=a*b;
System.out.println(" Multiplication =" + c);
break;
case 4:
c=a/b;
System.out.println(" Division =" + c);
```

```
break;
case 5:
System.exit(0);;
}
}
}
}
```

The following will be the output:

Press 1 for Addition

Press 2 for Subtraction

Press 3 for Multiplication

Press 4 for Division

Press 5 for Exit

4

Division =1

Press 1 for Addition

Press 2 for Subtraction

Press 3 for Multiplication

Press 4 for Division

Press 5 for Exit

2

Subtraction =10

Press 1 for Addition

Press 2 for Subtraction

Press 3 for Multiplication

Press 4 for Division

Press 5 for Exit

1

Addition =50

Press 1 for Addition

Press 2 for Subtraction

Press 3 for Multiplication

Press 4 for Division

Press 5 for Exit

3

Multiplication =600

Press 1 for Addition

Press 2 for Subtraction

Press 3 for Multiplication

Press 4 for Division

Press 5 for Exit

5

The ?: operator

Popularly known as conditional operator is another way of selecting one statement based on the value of the expression. The general form of ?: operator is:

Expression Condition ? Expression 1 : Expression 2

Example

Value = (a>b) ? 10 : 20;

In the above example the expression a>b will be evaluated if it is true than Value will get a value of 10 otherwise it will store 20. The above expression can be written using if…else statement as:

```
if (a>b)

Value =10;

else

Value =20;
```

The below program illustrates the use of ?: operator:

```
public class Operator
```

```
{
public static void main(String arg[])
{
int salary;
int da=70;
int basic = 8000;
salary = (da>50)? (basic + 100) : (basic +50);
System.out.println("The Net Salary is" + salary);
}
}
```

We will get the following output:

The Net Salary is 8100

The for loop

As you may know from your previous programming experience, looping constructs are an important part of nearly any programming language. Java is no exception. In fact, the most versatile is the for loop. The simplest form of the for loop is shown here:

```
for(initialization; condition; iteration) statement;
```

OR

```
for(initialization; condition; iteration)
{
Block of statements;
}
```

In its most common form, the initializations portion of the loop sets a loop control variable to an initial value. The condition is a Boolean expression that tests the loop control variable. If the outcome of that test is true, the for loop continues to iterate. If it is false, the loop terminates. The iteration expression determines how the loop control variable is changed each time the loop iterates. Here is a short program that illustrates the for loop:

```
public class ForExample {
```

```
public static void main(String args[]) {
int i;
for(i = 0; i<10; i++)
System.out.println("This is i: " + i);
}
}
```

The above program will generates the following output:

This is i: 0

This is i: 1

This is i: 2

This is i: 3

This is i: 4

This is i: 5

This is i: 6

This is i: 7

This is i: 8

This is i: 9

In this example, i is the loop control variable. It is initialized to zero in the

initialization portion of the for. At the start of each iteration, the conditional test x < 10 is performed. If the outcome of this test is true, the println( ) statement is executed, and then the iteration portion of the loop is executed. This process continues until the conditional test is false.

The while loop

Like for loop, while loop is a Java construct used for looping. The major difference is that the initialization and increment/decrement statements are not within the braces as in case of for loop:

```
//Initialization

while (condition)

{

block of statements including

increment / decrement

}
```

The following Java programme makes use of the while loop to generate the same output as in the above program:

```
public class WhileExample {

public static void main(String args[]) {

int i;

i=0; // Initialization

while (i<10)
```

```
{
System.out.println("This is i: " + i);
i++; // Increment
}
}
}
```

The output for the above program is:

This is i: 0

This is i: 1

This is i: 2

This is i: 3

This is i: 4

This is i: 5

This is i: 6

This is i: 7

This is i: 8

This is i: 9

The do…while loop

The only difference between while and do…while loop is that the condition in do…while is checked at the end of the first execution of the statement or block of statements. That means do…while loop will execute the statement at least once even if the condition is false. The syntax is as under:

```
//Initialization
do {
block of statements including
increment / decrement
} while (condition);
```

The same program can be executed using the do…while loop as under:

```
public class ForExample {
public static void main(String args[]) {
int i;
i=0; // Initialization
do
{
```

```
System.out.println("This is i: " + i);

i++; // Increment

}while (i<10);

}

}
```

The above program will generates the following output:

This is i: 0

This is i: 1

This is i: 2

This is i: 3

This is i: 4

This is i: 5

This is i: 6

This is i: 7

This is i: 8

This is i: 9

The following examples on control structures will further illustrate the use of these controls in Java.

The following Java program generates the prime numbers from 1 to 100 using two for loops. The outer for loop controls the number that needs to be verified and the inner loop controls the number with which the actual number is divided:

```
public class PrimeExample {

public static void main(String args[]) {

int i,j,r=1;

System.out.println("The Prime Numbers from 1 to 100 are ");

for(i=2;i<100;i++)

{

for (j=2;j<i;j++)

{

r= i % j;

if (r==0)

break;

}

if (r!=0)

System.out.println (i +"      ");

}

}

}
```

The above program will generates the following output:

The Prime Numbers from 1 to 100 are

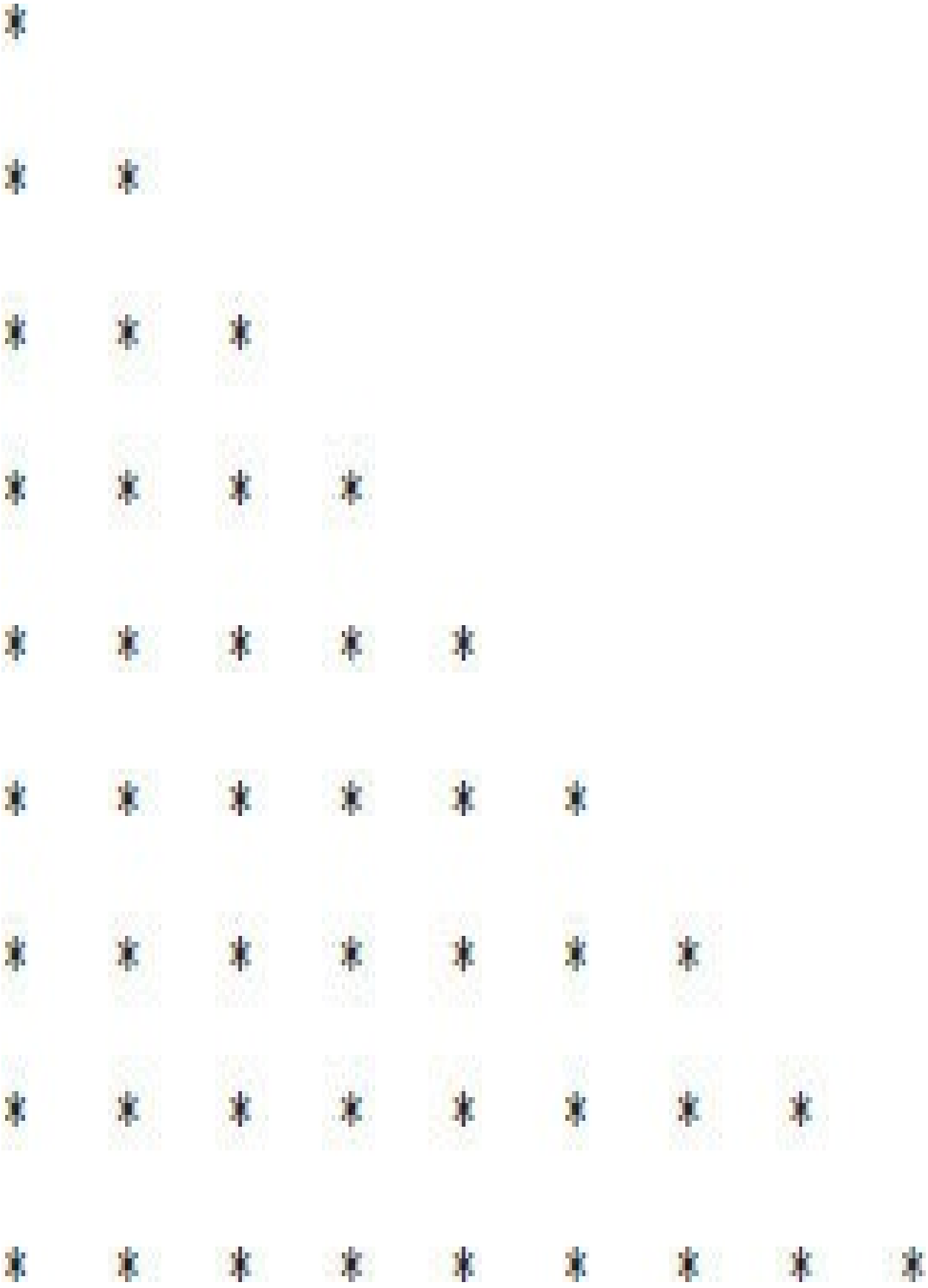
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

The following Java program makes use of nested while loops to generate the pattern:

```
public class LoopExample {

public static void main(String[] args) {

int i,j;

i=0;

while (i<10)

{

j=0;

while (j<i)

{

System.out.print("* ");

j++;

}
```

```
System.out.println("");
i++;
}
}
}
```

The above program will generates the following output:



The following Java program generates the Fibonacci series:

```
public class FibonacciSeries {
public static void main(String[] args) {
int a,b,c;
a=0;
b=1;
c=0;
System.out.println("The Fibonacci Series is as Under " );
do
{
System.out.print(" " + c);
a=b;
b=c;
c=a+b;
}
while (c<100);
}
}
```

The above program will generates the following output:

The Fibonacci Series is as Under

0 1 1 2 3 5 8 13 21 34 55 89

Command line arguments

If we want to input data to our program at run time, that is, at the time of execution, we can do that using the command line arguments. The input is the array of strings stored in arg[]. For example, the below mentioned program segment can store the arguments as under:

```
// Example program for Command Line Arguments

public class CmdLine

{

public static void main (String arg[])

{

if (arg.length ==0)

System.out.println("No Command Line Arguments are supplied as count = "+arg.length);

else

System.out.println("Total Number of Arguments = "+arg.length);


for(int i=0;i<arg.length;i++)

{

System.out.println( " The Arguments are" + arg[i]);

}
```

```
}
}
```

If the above program is executed as:

java CmdLine I Love Java Language

The above program will generates the following output:

Total Number of Arguments = 4

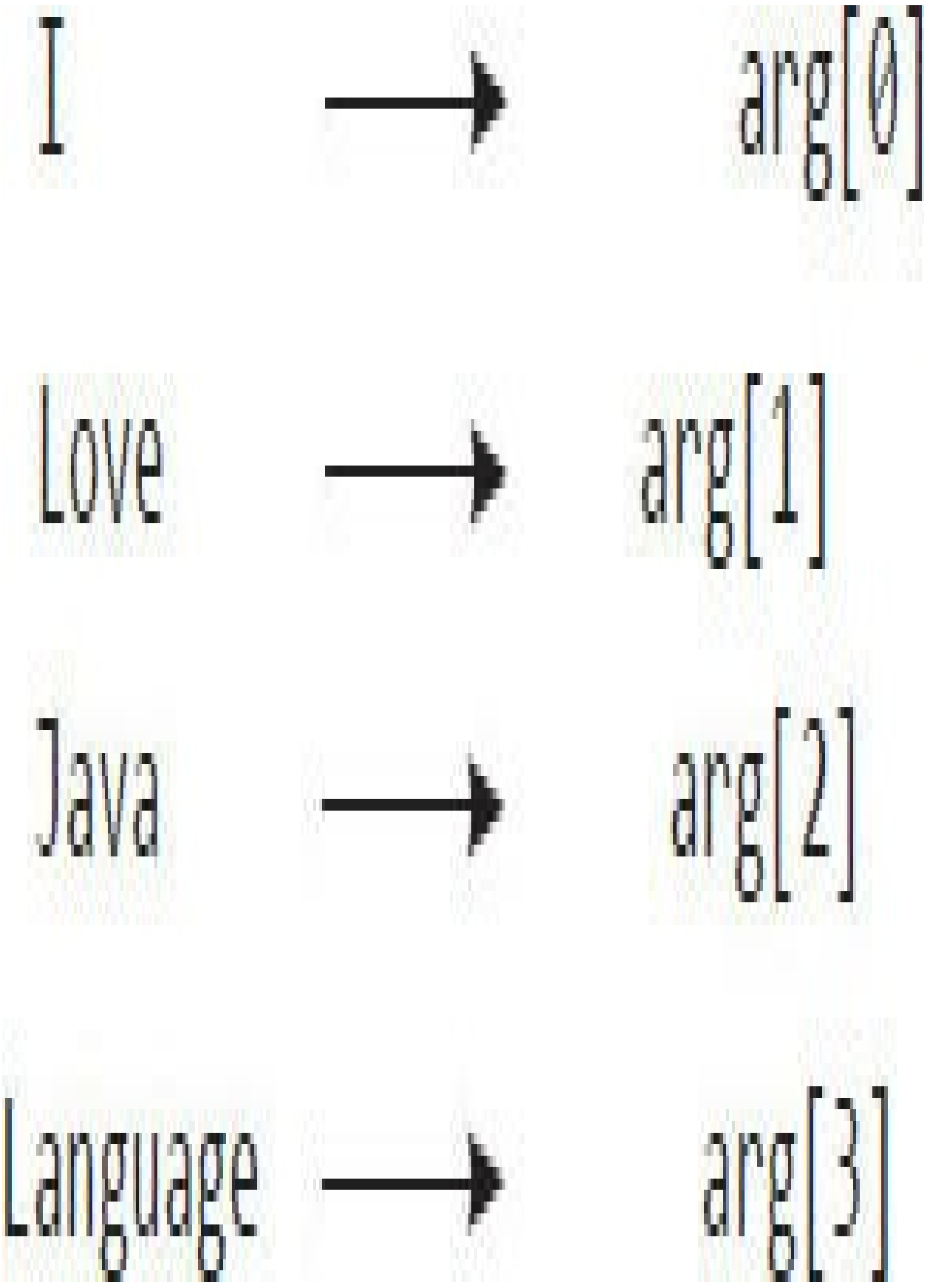
The Arguments are I

The Arguments are Love

The Arguments are Java

The Arguments are Language

In the above program we may say that the arguments are stored as:


I
arg[0]
Love
arg[1]
Java
arg[2]
Language
arg[3]

The length of the array depends on the number of arguments supplied. If for example the above program is executed with no arguments as:

java CmdLine

Then the output of the above program would be as:

No Command Line Arguments are supplied as count = 0

Here are some more programs on command line arguments. The below mentioned program gets the strings from the command line converts them into integers by making use of one of the wrapper class called as Integer. The execution steps are as under:

```
class CommandDemo

{

public static void main(String arg[])

{

int a[] = new int [10];

int sum = 0;

for(int i=0;i<arg.length;i++)

{
```

```
a[i]= Integer.parseInt(arg[i]);

sum = sum+ a[i];

}

System.out.println("Sum = " + sum);

}

} //End of the program
```

javac CommandDemo.java // for compilation

java CommandDemo 30 50 20 10 // supplying the command line arguments at the time of execution

The above program will generates the following output:

Sum = 110

In this example of command line arguments we get string from the command line and sort them. The sequence of steps is as under:

```
class StringSort

{

public static void main (String arg[])

{

int i,j;
```

```
for(i=0;i<arg.length;i++)
{
for(j=i+1;j<arg.length;j++)
{
if ( arg[i].compareTo(arg[j])>0)
{
String temp = arg[i];
arg[i]= arg[j];
arg[j] = temp;
}
}
}
System.out.println( " The Sorted Names are");
for(i=0;i<arg.length;i++)
System.out.println(arg[i]);
}
} // End of the Program
```

javac StringSort.java // for Compilation

java StringSort Dr Muneer Ahmad Dar // with command line arguments

After execution with above command line arguments the output would be:

The Sorted Names are

Ahmad

Dar

Dr

Muneer

In the following Java program the Armstrong number is checked:

```
public class ArmStrong
{
public static void main(String [] args)
{
int num=15051,r,sum=0;
int num2=num;
while(num>0)
{
r=num%10;
sum=(sum*10)+r;
num/=10;
}
```

```
if(sum==num2){
System.out.println("Armstrong number");
}
else{
System.out.println("Not ArmStrong number");
}
}
}
```

The above program will generates the following output:

Armstrong number

The following program generates the multiplication table of 7:

```
public class Mul
{
public static void main(String arg[])
{
int n=7;
for(int i=1;i<=10;i++)
```

```
{
System.out.println(n + "*" + i + " =" + (n*i));
}
}
} // End of the program
```

The above program will generates the following output:

7 * 1= 7

7 * 2= 14

7* 3= 21

7 * 4= 28

7 * 5= 35

7 * 6= 42

7 * 7= 49

7 * 8= 56

7 * 9= 63

7 * 10= 70

The following Java program calculates the sum of numbers:

```
public class Sum
{
public static void main(String [] args)
{
int num=1274,r,sum=0;
while(num>0)
{
r=num%10;
sum+=r;
num/=10;
}
// The sum is printed here
System.out.println("The Sum is"+sum);
}
}
```

The above program will generates the following output:

The Sum is 14

The following Java program calculates the reverse of a number:

```
public class Reverse
{
public static void main(String [] args)
{
int num=12345,r,reverse=0;
int num2=num;
while(num>0)
{
r=num%10;
reverse=(reverse*10)+r;
num/=10;
}
System.out.println("The reverse of "+num2+" is "+ reverse);
}
} // End of the Program
```

The above program will generates the following output:

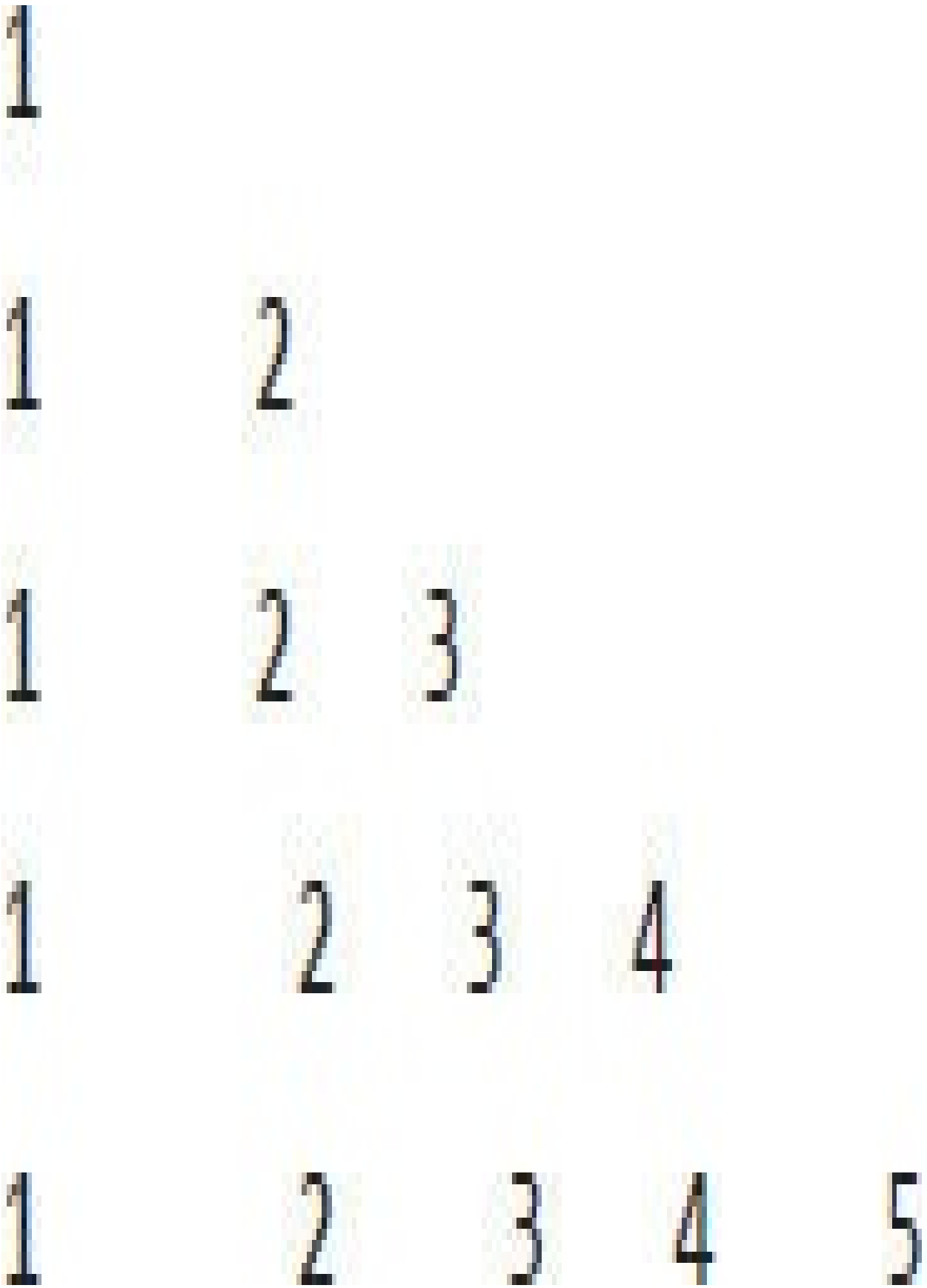
The reverse of 12345 is 54321

Assignment

Explain the use of various keywords used in public static void main(Stringarg [ ])?

What are the rules in Java to create a variable name?

Write a Java program to display the following output using the do…while loop.


1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

Explain the structure of a Java program? What are the mandatory sections in a Java program?

Using the command line arguments, get a number from a user and check whether a number is a prime number or not?

Input a string through command line and display the reverse of a given string?

CHAPTER 3

Object-Oriented Programming (OOP) in Java

Introduction

The basic building blocks of Java are the classes and objects. Right from the beginning, we have used classes and they define the inner details of any of the objects. They form the backbone of object-oriented programming in Java. If we want to implement any of the features in Java, be it creating a graphical user interface or dealing with input/ output it's only possible through classes and objects. If we are able to understand the concepts of objects and classes, then only we can understand the Java language properly and would be able to write some high-level programs as everything in Java is implemented through objects and classes.

Structure

Introduction

Creating a class and objects

Constructors in Java

Understanding the this keyword

Understanding access modifiers

Method overloading

Static methods

Inheritance

Understanding super, final, and abstract keywords

Assignments

Objective

This chapter explains the object oriented concepts of Java programming. After reading this chapter, a reader will be able to write the programs with objects and classes that are the backbones of Java program.

Introduction

The class is a template for its objects and we call object as an instance of that class. In fact, the most significant thing to recognizeregarding a class is that it defines a new data type. After creation, this new type can be used to generate objects of that kind. Therefore, a class is anoutline for an object, and an object is an instance of a class. We can say that everything in this universe is in term of objects and classes. For example we all are the instances of human class and if we further want to go ahead we may say that I belong to male or female class based on some specialized characteristics of an object. The classes have been used in the previous chapters but the classes and objects will be discussed with more detail in this chapter.

Broadly we can say that a class is a combination of data and methods. The variables, defined inside a class are called instance variables. The code is controlled within methods. Jointly, the methods and variables defined inside a class are called members of the class. In the majority of classes, the instance variables are acted upon and accessed by the methods defined for that class. Therefore, as a broad rule, it is the methods that decide how a class' data can be accessed. Variables defined inside a class are called instance variables because each instance of the class (that is, every object of the class) contains its individual reproduction of these variables. Therefore, the data intended for one object is disconnected and exclusive from the data for another object:

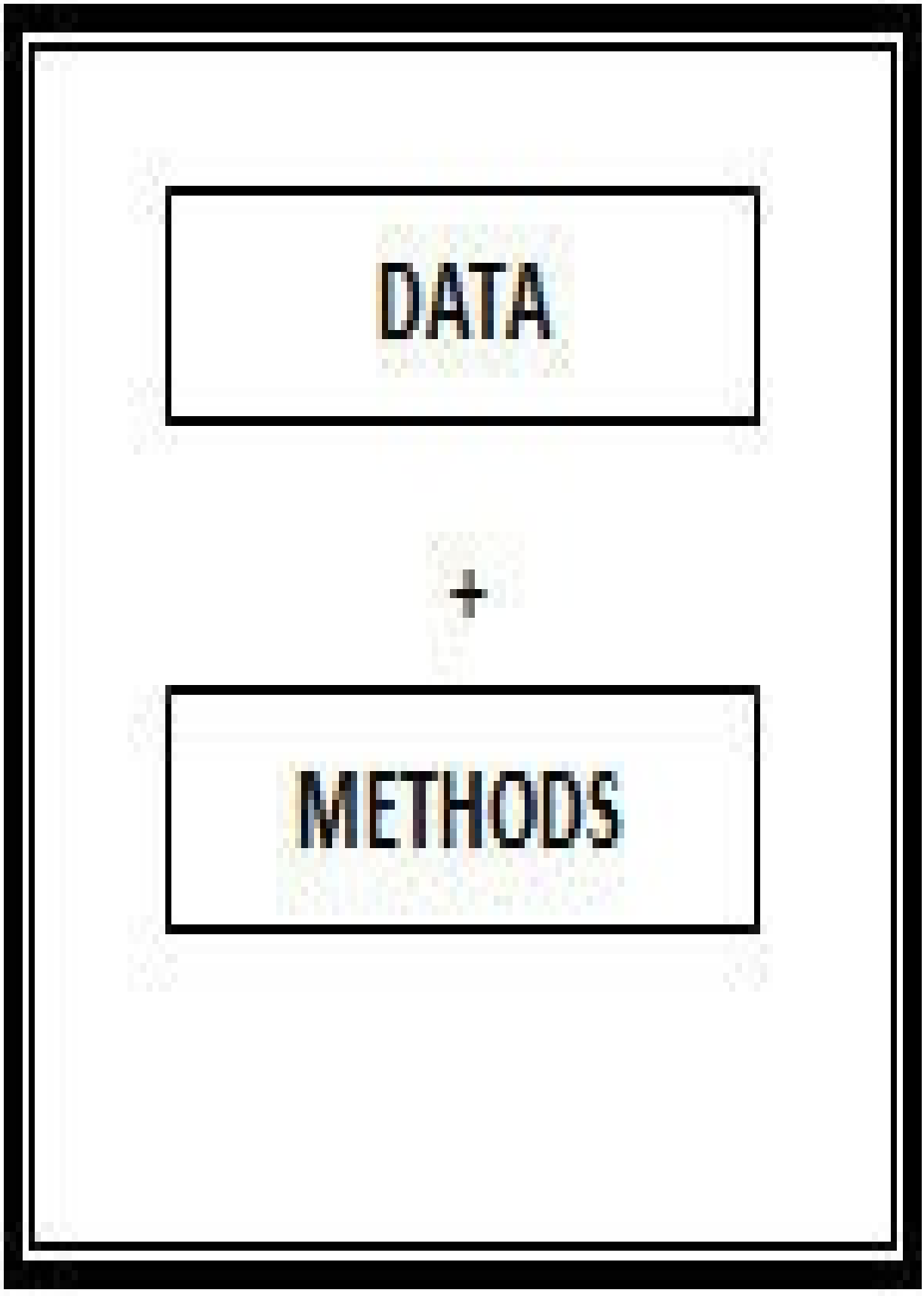

DATA
+
METHODS

Figure 3.1: Aclass with data and methods

Creatinga class and object

As already discussed, the class is a template and whatever we want to include in the class that will get reflected in its corresponding objects The general form of a class is as under:

class MyClass [extends classname, Implements interfacename]

{

// Data to be included in the class

[Data_Type variable_1;]

[Data_Type variable_2;]

[Data_Type variable_N;]

// Methods to be included in the class

[Data_Type method_name_1(Parameters)

{

body 1

}]

[Data_Type method_name_2(Parameters)

{

body 2

}]

[Data_Type method_name_3(Parameters)

{

body 3

}]

} // closing of class

Everything within square [] brackets is optional that means we can create a class without data and without methods. The example of an empty class is as under:

class Empty

{

}

The above class definition does nothing as it does not have any data nor any methods or member functions. The below examples will illustrate the use and creation of classes and objects.

In this example we are going to create a Student class and create its objects. We will further enhance this class in upcoming examples to understand the concept in a better way. The Student class is incorporated with three data members namely en_no, name and marks and two method or member functions namely get_Data() and put_Data(). It is pertinent to mention here that we have created two classes – Student and MainClass, the execution shall always start from the main method and we have included main method in MainClass.

We have created two objects of Student class by making use of new operator. We can create an instance or object of Student class with two ways as under:

```
Student s1 = new Student();
```

OR

```
Student s1;
s1=new Student();
```

After creating the objects, we can access the data members or the methods by making use of '.' (DOT) operator. For example, if we want to access the name of s1, we can uses1.name. Till now we have not used the access modifiers and as such the default access modifier friendly (not a keyword) is used by default. As the default access modifier of C++ is private, that is not the case with Java and the default access modifier of Java is friendly:

```
class Student
{
//Data
int en_no, marks;
String name;
// Methods
void getData(int no, int m,String nm)
```

```
{
en_no=no;
marks=m;
name=nm;
}
void putData()
{
System.out.println("Enrolment Number = " + en_no + "Name = " +name+
"marks =" + marks );
}
}

//Main Class
public class MainClass {
public static void main(String[] args) {
// TODO Auto-generated method stub
Student s1=new Student();
Student s2=new Student();
s1.getData(33, 500, "Abdul Wahid Wali");
s2.getData(13, 400, "Mohammad Saleem Mir");
s1.putData();
```

```
s2.putData();

}

}
```

We will get the following output:

Enrolment Number = 33Name = Abdul Wahid Walimarks =500

Enrolment Number = 13Name = Mohammad Saleem Mirmarks =400

We can create N number of objects in the same program. We will further enhance the above program and include one more class Box. The objects of Box are created and are accessed from the main class:

```
class Box

{

//Data

int length, breadth, height;

// Methods

void getData(int a, int b,int c)

{

length=a;

breadth=b;
```

```
height=c;

}

void putData()

{

System.out.println("Length = " + length + "breadth = " +breadth+ "height =" +
height );

}

}

class Student

{

//Data

int en_no, marks;

String name;

// Methods

void getData(int no, int m,String nm)

{

en_no=no;

marks=m;

name=nm;

}

void putData()
```

```
{

System.out.println("Enrolment Number = " + en_no + "Name = " +name+
"marks =" + marks );

}

}


//Main Class

public class MainClass {

public static void main(String[] args) {

// TODO Auto-generated method stub

Student s1=new Student();

Student s2=new Student();

Box b1 = new Box();

Box b2= new Box();

s1.getData(33, 500, "Abdul Wahid Wali");

s2.getData(13, 400, "Mohammad Saleem Mir");

b1.getData(2, 3, 9);

b2.getData(4, 7, 5);

s1.putData();

s2.putData();

b1.putData();
```

```
b2.putData();
}
}
```

We will get the following output:

Enrolment Number = 33Name = Abdul Wahid Walimarks =500

Enrolment Number = 13Name = Mohammad Saleem Mirmarks =400

Length = 2breadth = 3height =9

Length = 4breadth = 7height =5

In the above programs, the two classes namely Student and Box defines a new type of data. We use these names to create objects of type Student and Box. It is significant to keep in mind that a class declaration simply creates a template that can be used to define objects and it does not create an actual object.

In order to create a Student and a Box objects, we need to use a statements as under:

```
Student s1=new Student(); // Create a Student type object
Student s2=new Student(); // Create a 2nd Student type object
Box b1 = new Box(); // Create a Box type object
Box b2= new Box(); // Create a 2nd Box type object
```

After this statement executes, s1 and s2 will be an instance of Student class and b1 and b2 will be the objects of Box class. Every time we declare an instance of a class, we are creating an object that contains its individual copy of all instance variable defined by the class. Thus, every Student object willcontain its own copies of the instance variablesen_no, marks and name and every Box object will have its own copies of length, breadth and height as declared in their corresponding classes. To access these variables, you will use the dot (.) operator. The dot operator links the name of the object with the name of an instance variable. For example, to assign the marks variable of s1the value 50, we would use the following statement:

```
s1.marks = 50;
```

This statement tells the compiler to assign the copy of marks that is contained within the s1object the value of 50. In general, you use the dot operator to access both the instance variables and the methods within an object.

Therefore, if we created two objects of Student class and two objects of Box class, all the four objects will have the individual copy of their variables as declared in their corresponding classes:

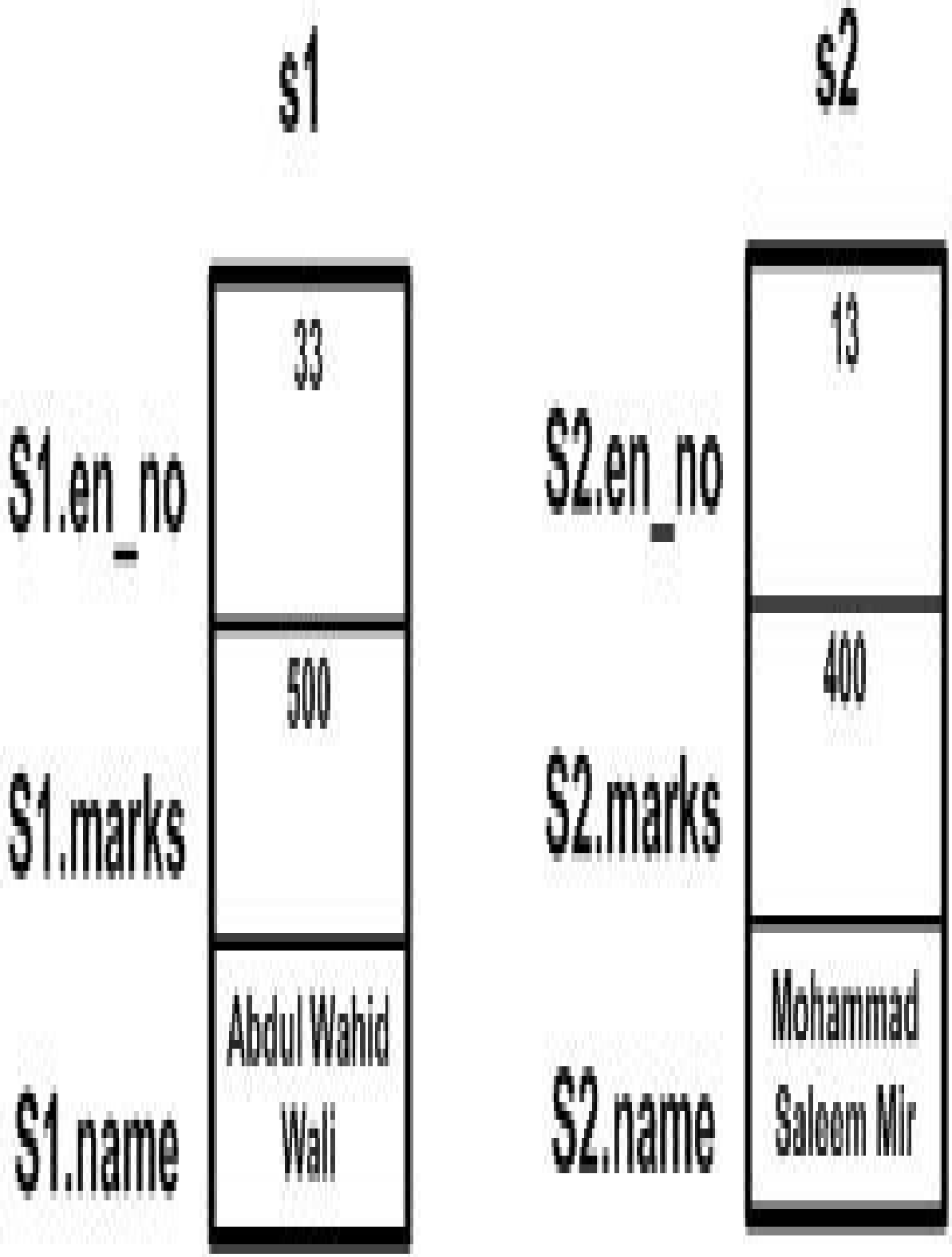

s1
s1.en_no
33
s1.marks
500
s1.name
Abdul Wahid
Wali
s2
s2.en_no
13
s2.marks
400
s2.name
Mohammad
Saleem Mir

Figure 3.2: Two objects of Student class

In the following Java program two objects of Rectangle class are created and the area function is invoked to display their area:

```
package book;

class Rectangle

{

int l,w;

void getData(int a, int b)

{

l=a;

w=b;

}


int area()

{

return(l*w);

}

}
```

```
public class ClassDemo {

public static void main(String[] args) {

Rectangle r1 = new Rectangle();

Rectangle r2 = new Rectangle();

r1.getData(20,30);

r2.getData(10,50);

System.out.println("Area of First Rectangle = " + r1.area());

System.out.println("Area of 2nd Rectangle = " + r2.area());

}

}
```

We will get the following output:

```
Area of First Rectangle = 600

Area of 2nd Rectangle = 500
```

A more practical example is implemented by creating a Stack. As we know a stack is a data structure thatperforms LIFO (last in first out) operation. A String type array is created within the Stack class and two instances of Stack class are created. Both the objects have a separate copy of String array and perform the push and pop operation:

```
package book;

class Stack {
```

```
String stck[] = new String[5];
int tos =-1;
void push(String item) {
if(tos==5)
System.out.println("Stack Overflow.");
else
stck[++tos] = item;
}
String pop() {
if(tos < 0) {
System.out.println("Stack underflow.");
return "";
}
else
return stck[tos--];
}
}
class StackExample {
public static void main(String args[]) {
Stack stack1 = new Stack();
Stack stack2 = new Stack();
```

```
//Push Elements in 1st Stack
stack1.push("Apple");
stack1.push("Ball");
stack1.push("Cat");
stack1.push("Dog");
stack1.push("Egg");
// Push Elements in 2nd Stack
stack2.push("Allah");
stack2.push("Basket");
stack2.push("Carrot");
stack2.push("Donkey");
stack2.push("Email");
System.out.println("Stack in mystack1:");
for(int i=0; i<5; i++)
System.out.println(stack1.pop());
System.out.println("Stack in mystack2:");
for(int i=0; i<5; i++)
System.out.println(stack2.pop());
}
}
}
```

We will get the following output:

Stack in mystack1:

Egg

Dog

Cat

Ball

Apple

Stack in mystack2:

Email

Donkey

Carrot

Basket

Allah

Constructors in Java

Constructors are the special member functions used to initialize the objects when they are created. The process to initialize all of the variables in a class all the time an instance is produced can be a tiresome activity. Even when we add ease functions like setData( ), it would be uncomplicated and extrabrief to include all of the arrangementsfinished at the instance the object is initiallyproduced. Since the necessity for initialization is so frequent, Java permits objects to initialize themselves while they are formed. This automatic initialization is carried outfrom beginning to end by the application of a constructor.

A constructor initializes an object instantly upon formation. It has the identical name as the class in which it is located and is similar to a method but we call it a special member function as it has some special features like the constructor is automatically called straight away after the object is formed, prior to the new operator completes. Constructors appear a slightweird and wonderful because they have no return type, not even void. This is because the inherent return type of a class' constructor is the class type itself.

Being a special member function of a class it has a set of specialized features as under:

Constructor has a same name as that of a class name. For example if we want to create a class Student, its constructor will have a same name as Student(); for example the below class has two constructors with the same name as class name Employee:

```
class Employee

{

// Default Constructor

Employee( )

{

// Body of the Constructor

}

// Parameterized Constructor

Employee( int d1, int d2)

{

// Body of the Constructor

}
```

Constructor does not return any value, not even void. As in the above example the constructor has no return type.

Constructors are invoked/called automatically when the object is created. If we have created a Box class and we are creating an object as Box b = new Box(); then it will call the Box()constructor automatically, if it exists. The above constructors are automatically invoked by the following statements:

```
Employee emp1= new Employee( ); // Will invoke the default constructor

Employee emp2= new Employee(2, 200 ); // Will invoke the constructor with
```

arguments

The below programs on constructors will clarify the concept properly.

In the following example we created a class Student with three data members. Now in order to initialize these data members we used the putData() member function in the previous example. Now as the objects are created we may initialize them by making use of constructor function. As such we replaced putData() function with Student() constructor with three objects to initialize the objects:

```
class Student

{

//Data

int en_no, marks;

String name;

// Constructor

Student(int no, int m,String nm)

{

en_no=no;

marks=m;

name=nm;

}
```

```
void putData()

{

System.out.println("Enrolment Number = " + en_no + "Name = " +name+
"marks =" + marks );

}

}

public class Constructor_Demo {

public static void main(String[] args) {

Student s1=new Student(3,300,"Arshid");

Student s2=new Student(4,500,"Rahid");

s1.putData();

s2.putData();

}

}
```

We will get the following output:

Enrolment Number = 3Name = Arshidmarks =300

Enrolment Number = 4Name = Rahidmarks =500

In the following example we created multiple constructors with different arguments and the default argument without any parameters. This concept is

called as constructor overloading. Now it depends on the object that is being created with the number and type of parameters. If the object being created has no argument, it will go to the default constructor with no arguments. The below example creates three constructors depending upon the parameters passed. The first object is created with default constructor and others with parameterized constructors:

```
class Student

{

//Data

int en_no, marks;

String name;

// Constructors

Student()

{

System.out.println("Default Constructor Invoked" );

}

Student(int r)

{

en_no =r;

System.out.println("Constructor with One Argument" );

}
```

```
Student(int no, int m,String nm)

{

en_no=no;

marks=m;

name=nm;

System.out.println("Constructor with three Arguments" );

}

void putData()

{

System.out.println("Enrolment Number = " + en_no + "Name = " +name+
"marks =" + marks );

}



}

public class Constructor_Demo {

public static void main(String[] args) {

Student s1=new Student();

Student s2=new Student(44);

Student s3=new Student(4,500,"Rahid");

}

}
```

We will get the following output:

Default Constructor Invoked

Constructor with One Argument

Constructor with three Arguments

The Stack program is repeated with the usage of constructors:

```
package book;
class Stack {
String stck[] = new String[5];
int tos ;
// Constructor
Stack( )
{
System.out.println(" Constructing a Stack:");
tos = -1
}
void push(String item) {
if(tos==5)
```

```
System.out.println("Stack Overflow.");
else
stck[++tos] = item;
}
String pop() {
if(tos < 0) {
System.out.println("Stack underflow.");
return "";
}
else
return stck[tos--];
}
}
class StackExample {
public static void main(String args[]) {
Stack stack1 = new Stack();
Stack stack2 = new Stack();
//Push Elements in 1st Stack
stack1.push("Apple");
stack1.push("Ball");
stack1.push("Cat");
```

```
stack1.push("Dog");
stack1.push("Egg");
// Push Elements in 2nd Stack
stack2.push("Allah");
stack2.push("Basket");
stack2.push("Carrot");
stack2.push("Donkey");
stack2.push("Email");
System.out.println("Stack in mystack1:");
for(int i=0; i<5; i++)
System.out.println(stack1.pop());
System.out.println("Stack in mystack2:");
for(int i=0; i<5; i++)
System.out.println(stack2.pop());
}
}
}
```

We will get the following output:

Constructing a Stack

Constructing a Stack

Stack in mystack1:

Egg

Dog

Cat

Ball

Apple

Stack in mystack2:

Email

Donkey

Carrot

Basket

Allah

Understanding the this keyword

The only pointer available in Java is this pointer, which refers to the current object which is invoked. Every now and then a function will require to refer to the object that invoked or called it. To permit this, Java defines the this keyword. this can be used within every method to refer to the current object. That is, this is at all times a reference to the object on which the method was invoked. Say for example we are in a function display() of any class Box with volume as data member then within display if we want to get the value of volume, we can get that using these two forms:

```
void display()

{

System.out.println("Volume=" + volume );

}
```

OR

```
void display()

{

System.out.println("Volume=" +this. volume );

}
```

So to access the data members, the this operator can be used as in the following class the data members are accessed using the this keyword:

```
class Student

{

int en_no, marks;

String name;

Student( )

{ }

Student (int a, int b, String nm)

{

this.en_no = a;

this.marks = b;

this.name=nm;

}

}
```

The following programs will make use of this to have more understanding.

In the following Java program creates a Box class and creates three instances of Box class. We compared the volume of two Box objects so as to store the larger in the third object. It was not possible to access the invoking object b1 as its scope is within the braces it is created, so we made use of this operator to access

the object. We also returned this object which refers to the object which has invoked the function:

```
class Box

{

private int l,b,h, vol;

Box()

{

l=0;

b=0;

h=0;

}

Box(int a,int b, int c)

{

l=a;

this.b=b; // this used to remove ambiguity

h=c;

vol= a*b*c;

}

void display()

{
```

```
System.out.println( "Volume of Larger Box = " + vol);
}
Box compare(Box b2) {
if (this.vol>b2.vol)
return this;
else
return b2;
}

public static void main(String[] args) {
Box b1= new Box(2,3,4);
Box b2= new Box(12,3,40);
Box temp = new Box();
temp=b1.compare(b2);
temp.display();
}
}
```

We will get the following output:

Volume of Larger Box = 1440

In the following example we are going to extend the previous program and add two objects of Box class to store the result in the third object. The this operator is used to add the contents of the invoking object:

```
class Box

{

private int l,b,h, vol;


Box()

{

l=0;

b=0;

h=0;

}

Box(int a,int b, int c)

{

l=a;

this.b=b; // this used to remove ambiguity

h=c;

vol= a*b*c;

}
```

```
void display()
{
System.out.println( "Volume After Addition = " + vol);
}
Box compare(Box b2) {
if (this.vol>b2.vol)
return this;
else
return b2;
}
Box add(Box b2)
{
return (new Box(this.b+b2.b,this.h+b2.h,this.l+b2.l));
}

public static void main(String[] args) {
// TODO Auto-generated method stub
Box b1= new Box(2,3,4);
Box b2= new Box(12,3,40);
Box temp = new Box();
temp=b1.compare(b2);
```

```
//temp.display();

temp=b1.add(b2);

temp.display();

}

}
```

We will get the following output:

Volume After Addition = 3696

In the following Java program, the this operator is used to access the invoking object to compare it with the other object. If the invoking object is larger, the this is returned that refers to the current object:

```
package book;

public class Rect

{

int length,width;

Rect()

{

this.length=0;

this.width=0;
```

```
}
Rect(int a, int b)
{
this.length=a;
this.width=b;
}
int area()
{
return(length*width);
}
Rect max(Rect obj)
{
if (this.area() > obj.area())
return(this);
else
return(obj);
}
}
public class ThisOperatorDemo {
public static void main(String[] args) {
Rect r1 = new Rect(20,30);
```

```
Rect r2 = new Rect(10,500);
System.out.println("Area of First Rectangle = " + r1.area());
System.out.println("Area of 2nd Rectangle = " + r2.area());
System.out.println("Area of Largest Rectangle = " + r1.max(r2).area());
}
}
```

We will get the following output:

Area of First Rectangle = 600

Area of 2nd Rectangle = 5000

Area of Largest Rectangle = 5000

Understading the access modifiers

As we saw in previous programs that the data and methods are accessible from outside the class. This is not the right approach in OOP as we want to hide the data from outside the class. In order to have data encapsulation that is to hide the data so that it should be accessible only from within the same class, Java as in other object-oriented languages provides the access modifiers to restrict the access of data and methods. Java has three visibility controls public, private, and protected which can be used with the data members or member functions. Now in previous examples we have not used any of the access modifiers and still we are able to access the data and methods, the reason for that is Javacomes up with default access specifier and that is friendly whose scope is within all the classes of same packages (package is a group of similar classes. To be discussed in upcoming chapters).

public: If we want our data and methods to be available for all other classes within and outside package, then we can make use of public access control. Normally we want to hide our data and make member functions available from other classes so we use public with the methods so that they can be invoked from other classes. The syntax for public access control is as under:

```
public float data1;

public void method_name()

{

}
```

private: When a member of a class is specified as private, then that member can only be accessed by other members of its own class. This is the reason why main( ) has always been preceded by the public specifier. It is called by code that is outside the program—that is, by the Java run-time system. When no access specifier is used, then by default the member of a class is public inside its personal package, but cannot be accessed outside of its package.

protected: It is mainly used in inheritance, the data declared as protected can be accessed from its subclasses which are within or outside the package. If we want our data to be accessed by its sub classes from within the same package only, we can make use of private protected.

To understand the difference between public and private, the following program creates data_a as default, data_b as public, and data_c as private:

```
class TestData {

int data_a; // default access

public int data_b; // public access

private int data_c; // private access

// As the private data cannot be accessed directly this method returns the value of data_c

void setc(int value)

{

// set the value of data_c

data_c = value;

}
```

```
int getc() {

// get the value of data_c

return c;

}

}

class AccessTestDemo {

public static void main(String args[]) {

TestData ob = new TestData();

// These are OK, data_a and data_b may be accessed directly

ob.data_a = 10;

ob.data_b = 20;

// This is not OK and will cause an error as data_c is private and can be accessed
only //within the class

ob.data_c = 100; // Error!

//We must access data_c through its methods

ob.setc(100); // OK

System.out.println("data_a, data_b, and data_c: " + ob.data_a + " " +ob.data_b +
" " + ob.getc());

}

}
```

The below table lists the various access specifiers used inside a class, subclass,

and a package and from other locations within a Java project:

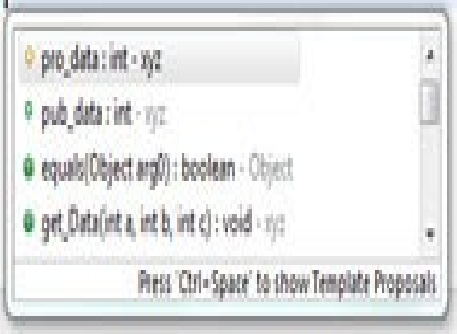
	public	protected	Friendly (Default)	private pr
Same class	yes	yes	yes	yes
Sub class within package	yes	yes	yes	yes
Other class within package	yes	yes	yes	No
Sub class outside package	yes	yes	No	yes
Non sub class outside package	yes	No	No	No

Table 3.1: Illustration of access modifiers

Now let us try to understand the visibility controls through programs.

In the following example we are using the access modifiers to check their visibility. We have defined three variables as public, protected and private. After creating an object of xyz class in Mc, we are not able to access the pri_data as its access modifier is private and private data is accessible only through the member functions of the same class:

```
2
3  class xyz {
4      public int pub_data;
5      private int pri_data;
6      protected int pro_data;
7      public void get_Data(int a,int b,int c)
8      {
9          pub_data=a;
10         pri_data=b;
11         pro_data=c;
12
13     }
14
15 }
16 class Mc
17 {
18     public static void main(String arg[])
19     {
20         xyz obj = new xyz();
21         obj.
22     }
23 }
24
25
```



In the following Java program, the Stack is implemented. The stck [] and tos is made private so that they are accessible only through push( ) and pop ( ) functions as these methods belong to the same class:

```
package book;

class Stack {

private String stck[] = new String[5];

private int tos =-1;

void push(String item) {

if(tos==5)

System.out.println("Stack Overflow.");

else

stck[++tos] = item;

}

String pop() {

if(tos < 0) {

System.out.println("Stack underflow.");

return "";

}

else

return stck[tos--];
```

```
}
}
class StackExample {
public static void main(String args[]) {
Stack stack1 = new Stack();
Stack stack2 = new Stack();
//Push Elements in 1st Stack
stack1.push("Apple");
stack1.push("Ball");
stack1.push("Cat");
stack1.push("Dog");
stack1.push("Egg");
// Push Elements in 2nd Stack
stack2.push("Allah");
stack2.push("Basket");
stack2.push("Carrot");
stack2.push("Donkey");
stack2.push("Email");
System.out.println("Stack in mystack1:");
for(int i=0; i<5; i++)
System.out.println(stack1.pop());
```

```
System.out.println("Stack in mystack2:");
for(int i=0; i<5; i++)
System.out.println(stack2.pop());
}
}
```

We will get the following output:

Stack in mystack1:

Egg

Dog

Cat

Ball

Apple

Stack in mystack2:

Email

Donkey

Carrot

Basket

Allah

Method overloading

If we have multiple methods with different parameters having same name to implement multiple functionalities based on their arguments, the concept is called as method overloading. Based on the name of the method and type and number of arguments, Java calls a particular method to implement the functionality. The process is called as polymorphism– poly means many and morphism means forms. The same concept is applicable to constructors also as we can have a default and parameterized constructors to implement the initialization of objects in different ways. When an overloaded method is invoked, Java uses the type and/or number of arguments as its channel to decide which version of the overloaded method to essentially call. Thus, overloaded methods have to be different in the type and/or number of their parameters. While overloaded methods may have different return types, the return type alone is inadequate to differentiate two versions of a method. When Java encounters a call to an overloaded method, it basically executes the version of the method whose parameters match the arguments used in the call. The following example illustrates the use of constructor and method overloading.

In this Java program creates two constructors with different arguments called as constructor overloading. The other three methods of show() with no argument, one and two arguments are created to give different functionalities:

```
public class Overloading_Demo {

public int a, b;

Overloading_Demo() {

a=0;

b=0;
```

```
}
Overloading_Demo(int x,int y)
{
a=x;
b=y;
}
void show()
{
System.out.println("Show Method with No Arguments");
}
void show(int a)
{
System.out.println("Show Method with One Argument");
}
void show(int a, int b)
{
System.out.println("Show Method with Two Arguments");
}

public static void main(String[] args) {
Overloading_Demo obj1 = new Overloading_Demo();
```

```
Overloading_Demo obj2 = new Overloading_Demo(2,5);

obj1.show();

obj1.show(1);

obj1.show(1,2);

}

}
```

We will get the following output:

Show Method with No Arguments

Show Method with One Argument

Show Method with Two Arguments

A more practical example of method overloading is to access the different methods based on the parameters. For example in the below program display() function is overloaded with one and two arguments. Both the functions are invoked with one and two arguments:

```
package book;

class Exam

{

int roll;
```

```
String name;
Exam()
{}
Exam(int a, String s)
{
roll=a;
name=s;
}
void display(int x)
{
System.out.print(roll);
}
void display(int x, int y)
{
System.out.print(roll + " " + name);
}
}

class Result extends Exam
{
int marks;
```

```
Result()
{}
Result(int a, String s, int m)
{
super(a,s);
marks = m;
}
void display()
{
super.display(1);
super.display(1,2);
System.out.println(" " +marks );
}
staticvoid sort(Result a[], int n)
{
Result temp = new Result();
for(int i=0;i<n;i++)
{
for(int j=(i+1);j<n;j++)
{
if(a[i].marks<a[j].marks)
```

```
{
temp = a[i];
a[i]= a[j];
a[j]=temp;
}
}
}
}
}
public class Overloading_Demo {
public static void main(String[] args) {
Result a[] = new Result[5];
a[0] = new Result(2,"Maheen", 200);
a[1] = new Result(3,"Hina", 210);
a[2] = new Result(4,"Mehak", 100);
a[3] = new Result(5,"Mareen", 150);
a[4] = new Result(6,"Sadaf", 120);
System.out.println("Before Sorting ");
for(int i=0;i<5;i++)
a[i].display();
Result.sort(a, 5);
```

```
System.out.println("After Sorting based on Marks and Sports marks");
for(int i=0;i<5;i++)
a[i].display();
}
}
```

We will get the following output:

Before Sorting

22 Maheen 200

33 Hina 210

44 Mehak 100

55 Mareen 150

66 Sadaf 120

After Sorting based on Marks and Sports marks

33 Hina 210

22 Maheen 200

55 Mareen 150

66 Sadaf 120

44 Mehak 100

The static keyword

If we want a data member of a class to be shared by all the objects of that class we can create that data member as static variable. A separate copy of data variable is created for all the objects who belong to that class as in our earlier examples. The same is the case with methods. Now to call these data members or member functions there is no need to create any object as these can be called by the name of the class itself. For example there are some built in static method in several in build classes of Java. Let us consider the declaration of parseInt method in Integer class of lang package.

static int parseInt(String arg)

This method converts a String integer into int. To call this static method, there is no need to create any object and it can be called by the class name itself as under:

int n = Integer.parseInt("300");

A class member (method or data) that will be used alone of any object of that class must be created as static. In general, a class member has to be accessed simply in combination with an object of its class. Yet, it is doable to build a member that can be used by itself, with nomention to anexplicit instance. To generate such a member, precede its declaration with the keyword static. When a member is declared static, it can be accessed prior to any objects of its class are produced, and without mention to any object. You can declare both methods and variables to be static. The most general example of a static member is main( ). The main( ) method is declared as static because it must be called before any

objects exist. Instance variables declared as static are, essentially, global variables. When objects of its class are declared, no copy of a static variable is made. Instead, all instances of the class share the same static variable. Methods declared as static have a number of boundaries:

They can only call other static methods.

They must only access static data.

They cannot refer to this or super in any way

In the following Java program, we have created a count as a static int variable to count the total number of objects created. One more static function called display is defined to return the value of count. We called the display function with the name of the class and it prints the total number of objects created:

```
//Java Program using static

public class Static_Demo {

static int count;

public Static_Demo() {

count++;

}
```

```
static int diplay_count()
{
return count;
}
public static void main(String[] args) {
Static_Demo obj1=new Static_Demo();
Static_Demo obj2=new Static_Demo();
Static_Demo obj3=new Static_Demo();
Static_Demo obj4=new Static_Demo();
Static_Demo obj5=new Static_Demo();
Static_Demo obj6=new Static_Demo();
System.out.println("Total Objects Created = " +Static_Demo.diplay_count());
}
}
```

We will get the following output:

Total Objects Created = 6

In the following Java program the static block is used As soon as the Static_Demo class is loaded, all of the static statements are run. First, a is set to 3, then the static block executes, which prints a message and then initializes b to

a * 4 or 12. Then main() is called, which calls meth(), passing 33 to x. The three println() statements refer to the two static variables a and b, as well as to the local variable x:

```
package book;

public class Static_Demo {

static int a = 3;

static int b;

static void st_Fun(int x) {

System.out.println("x = " + x);

System.out.println("a = " + a);

System.out.println("b = " + b);

}

static {

System.out.println("Initialization of static.");

b = a * 4;

}

public static void main(String args[]) {

st_Fun(33);

}

}
```

We will get the following output:

Initialization of static.

x = 33

a = 3

b = 12

Inheritance

One of the most exciting feature of OOPis inheritance that is reusing the existing code. If we have already written lot of code and we want to reuse that we can simply inherit it to our new class, this process of extending the base class features into the derived class is called as inheritance. To inherit a class, we just integrate the definition of one class into another by using the extends keyword.

The following program implements the basic inheritance. The BaseClass with data members as da and db along with the member function showBase is defined in the BaseClass. Now another class named as derived class is created and it extends the BaseClass which means that the data and methods of BaseClass are now available for the DerivedClass. In the main method the objects of both the classes are created to access their respective members along with the data and methods of the BaseClass.

```
package book;

class BaseClass {

int da, db;

void showBase() {

System.out.println("da = " + da + " db = " + db);

}

}

// Create a subclass by extending class BaseClass.

class DerivedClass extends BaseClass {
```

```
int dc;

void showdc() {
System.out.println("dc = " + dc);
}
void showMul() {
System.out.println("da*db*dc = " + (da*db*dc));
}
}

public class InheritanceExample {
public static void main(String arg[]) {
BaseClass superObj = new BaseClass();
DerivedClass derObj = new DerivedClass();
superObj.da = 10;
superObj.db = 20;
System.out.println("Contents of superObj: ");
superObj.showBase();;
System.out.println();

/*The subclass has access to all public members of its superclass. */
```

```
derObj.da = 7;
derObj.db = 8;
derObj.dc = 9;
System.out.println("Contents of subObj: ");
derObj.showBase();
derObj.showdc();
System.out.println();
System.out.println("Multiplication of da, db and dc in subOb:");
derObj.showMul();
}
}
```

We will get the following output:

Contents of superObj:

da = 10 db = 20

Contents of subObj:

da = 7 db = 8

dc = 9

Multiplication of da, db and dc in subOb:

da*db*dc = 504

In the above example, no access modifier is used although a subclass includes each and every one of the members of its superclass, it cannot access those members of the superclass that have been declared as private. For example, consider the following simple Java program:

The following program will not compile and will generate the error as the private data cannot be accessed from outside its class in which it is declared:

```
package book;

class FirstClass {

int fr_data; // As default

private int pr_data; // As private data

void setData(int a, int b)

{

fr_data = a;

pr_data = b;

}

}

class SecondClass extends FirstClass {

int total;

void sum() {

total = fr_data + pr_data;
```

```
// ERROR, pr_data not accessible
}
}

public class AccessExampleInheritance {
public static void main(String args[]) {
SecondClass subOb = new SecondClass();
subOb.setData(100, 120);
subOb.sum();
System.out.println("Total is " + subOb.total);
}
}
```

The above program is modified to access the private data defined in the FirstClass. The revised program is as under:

```
package book;
class FirstClass {
int fr_data; // As default
private int pr_data; // As private data
void setData(int a, int b)
```

```
{
fr_data = a;
pr_data = b;
}
int getPrData()
{
return pr_data;
}
}
class SecondClass extends FirstClass {
int total;
void sum() {
total = fr_data + getPrData();
// ERROR, pr_data not accessible
}
}

class AccessExampleInheritance {
public static void main(String args[]) {
SecondClass subOb = new SecondClass();
subOb.setData(100, 120);
```

```
subOb.sum();

System.out.println("Total is " + subOb.total);

}

}
```

We will get the following output:

Total is 220

Like C++ and many other object oriented programming languages, Java implements inheritance in many forms as detailed under.

Single inheritance

If we have a class A and its getting inherited by class B, we call class A as a base class and class B as derived class. The definition of these two classes is as under:

```
class A
{
// Data + Methods of Class A
}
class B extends B
{
// Data + Methods of Class B
}
```

The below diagram explains the single inheritance:

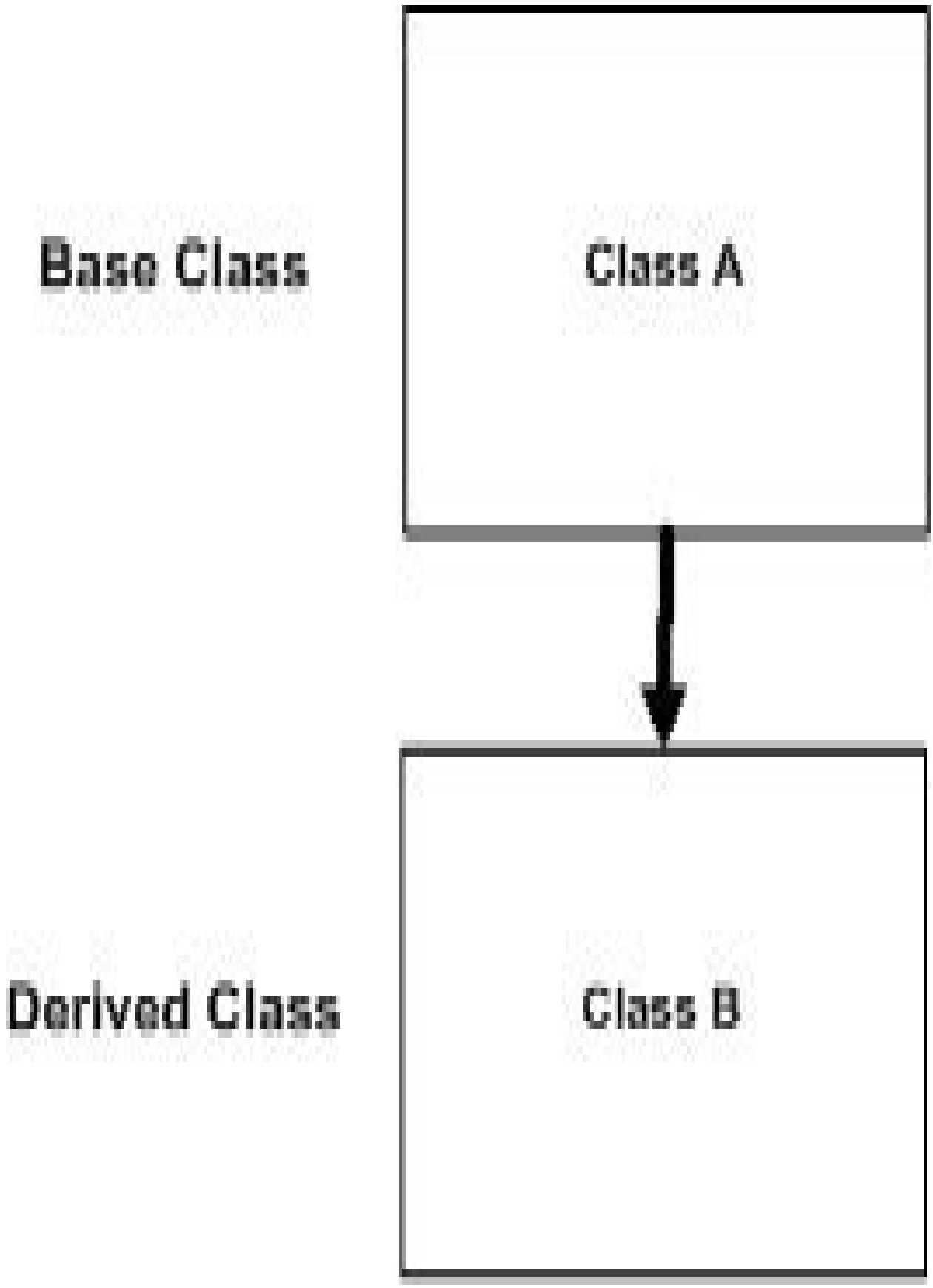

Base Class
Class A
Derived Class
Class B

Figure 3.3: Single inheritance

Hierarchical inheritance

As depicted by the diagram below, A single class is inherited by many derivedclasses. The definition of classes is as under:

```
class A
{
// Base Class
}
class B extends A
{
// Definition of class B
}
class C extends A
{
// Definition of class C
}
class D extends A
{
// Definition of class D
```

}

The below diagram explains the hierarchical inheritance:

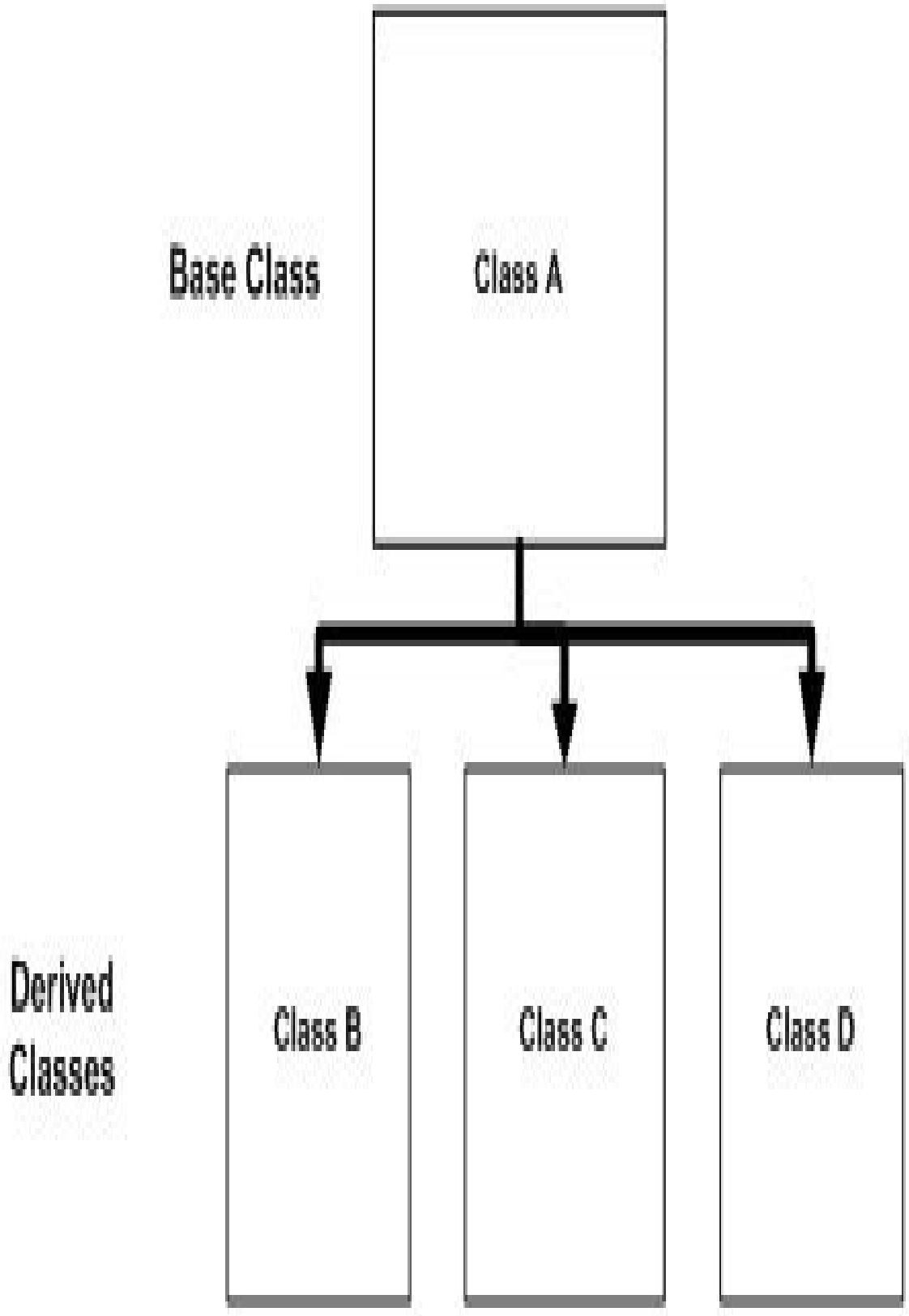

Base Class
Class A
Derived Classes
Class B
Class C
Class D

Figure 3.4: Hierarchical inheritance

Multilevel inheritance

In case of multilevel inheritance, a class is extended by one class and that extended class is further extended by another class. The last class will have the features of the first and the second base classes. The definition of three classes is as under:

```
class A

{

// Data + Methods of Base class

}

class B extends A

{

// Body of B class

}

class C extends B

{

// Body of C class

}
```

The below diagram explains the multilevel inheritance:

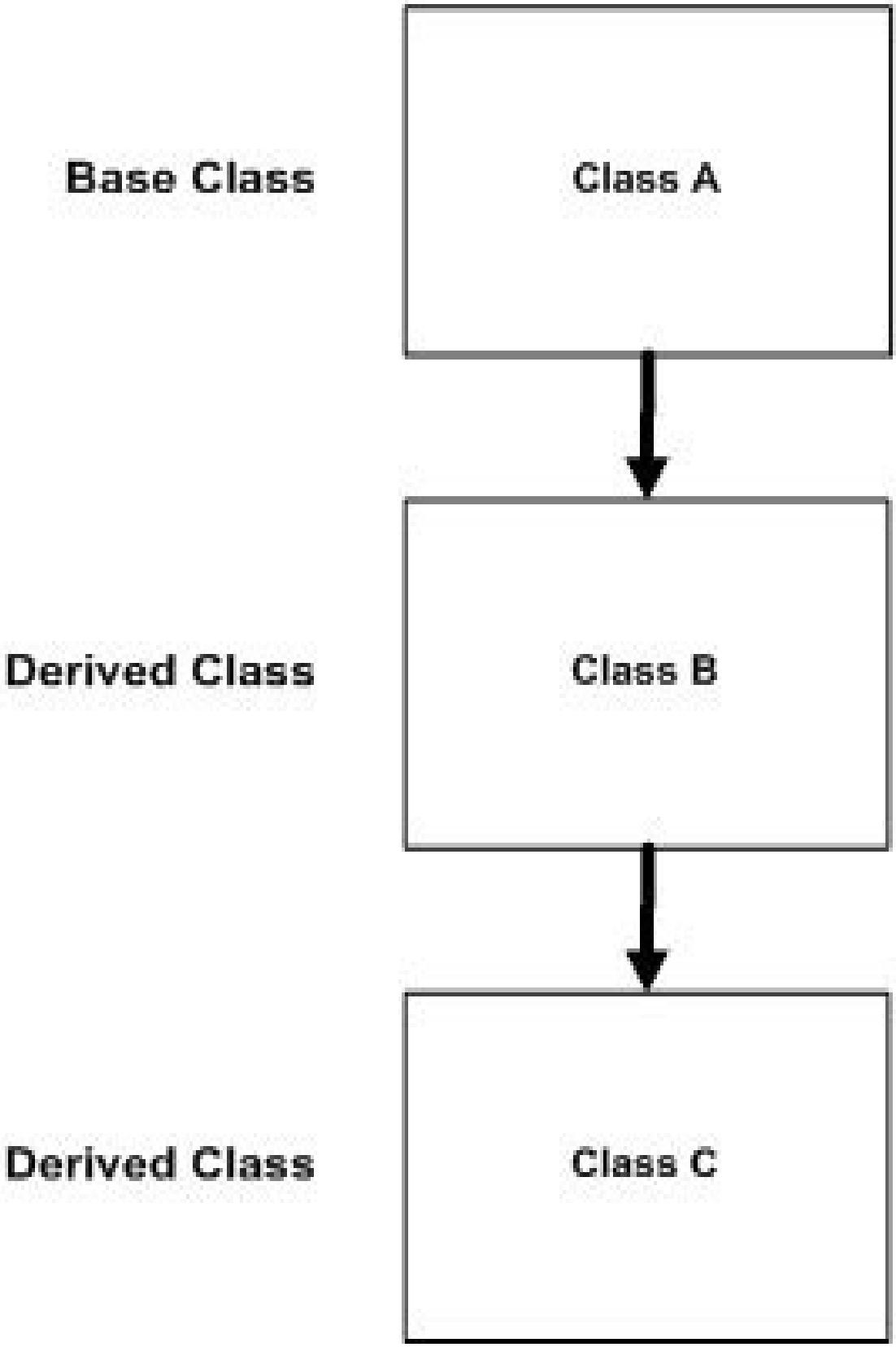

Base Class
Class A
Derived Class
Class B
Derived Class
Class C

Figure 3.5: Multilevel inheritance

Multiple inheritance

The multiple inheritance is not directly possible in Java. That means two classes cannot be inherited by a single class. The following statement is invalid in Java if A and B are two classes.

Class C extend A, B // Invalid

The other option for multiple inheritance in Java is through interfaces (To be discussed in next chapter). Now, the definition of the classes can be as under:

```
interface MyInterface
{
// Body of Interface that includes abstract methods and final data
}
class A
{
// Definition of class A
}
class B extends A implements MyInterface
{
```

```
// Body of class B
}
```

The below diagram explains the multiple inheritance:

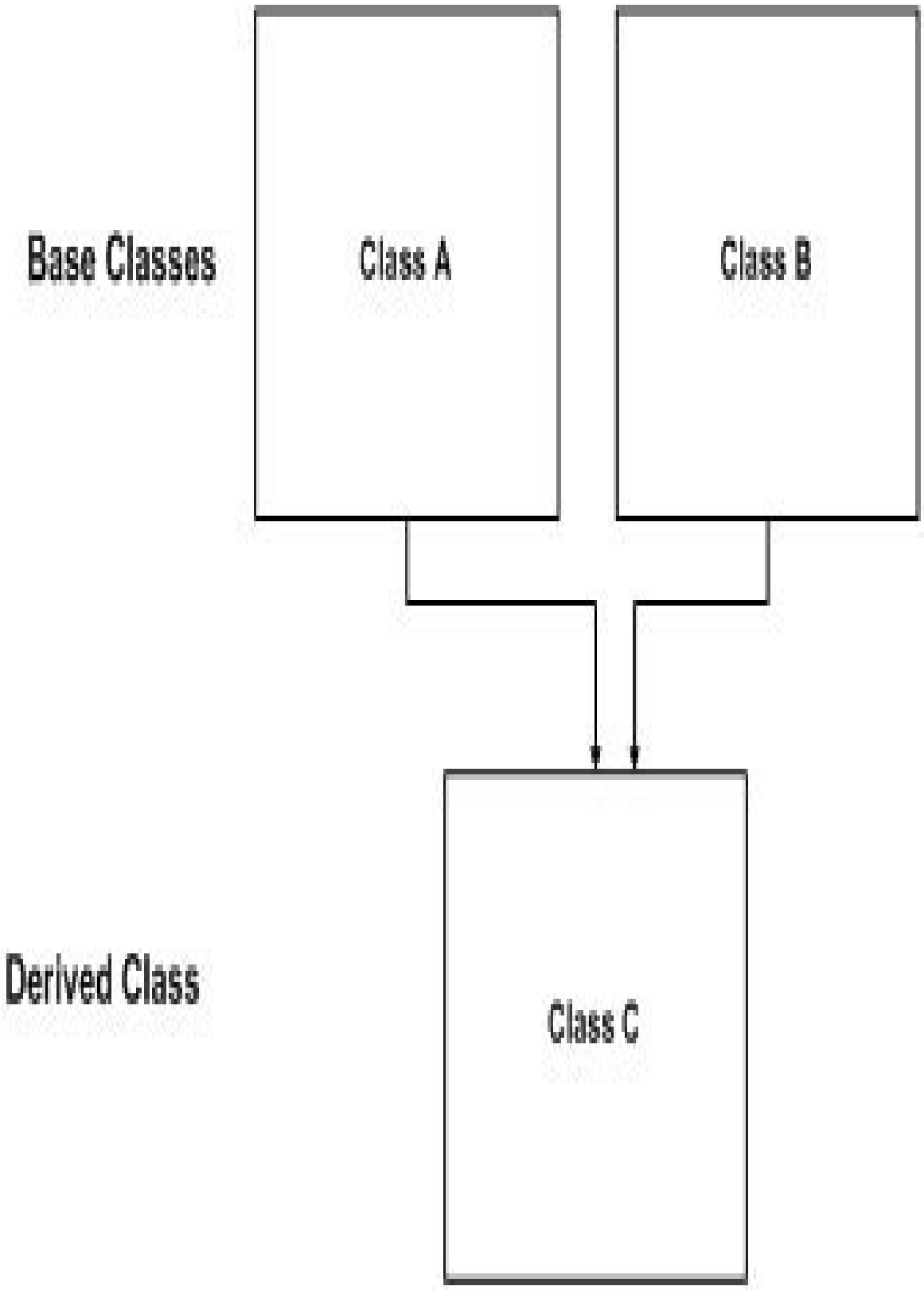

Base Classes
Class A
Class B
Derived Class
Class C

Figure 3.6: Multiple inheritance

In this program we have implemented single inheritance. Student1 is created as a base class and Result1 class extends all the features of base class:

```
class Student1 {

protected int roll,marks;

private String name;

Student1()

{

}

Student1(int a,int b,String nm)

{

roll=a;

marks=b;

name=nm;

}

void display()

{

System.out.println("Roll Number " + roll+" \nName "+ name +"\nMarks " +
marks);
```

```
}
}
public class Result1 extends Student1
{
private int sp_marks;
Result1()
{}
Result1(int a,int b,String s,int m)
{
super(a,b,s);
sp_marks=m;
}
int get_Total()
{
return(sp_marks+marks);
}
public static void main(String arg[])
{
Result1 s1 =new Result1(3,400,"Feroz",300);
s1.display();
System.out.println("Total Marks Are " + s1.get_Total());
```

```
}

}
```

We will get the following output:

Roll Number 3

Name Feroz

Marks 400

Total Marks Are 700

In the following program, we use single inheritance to sort the students data based on marks:

```
class Exam

{

int roll;

String name;

Exam()

{}

Exam(int a, String s)

{
```

```
roll=a;
name=s;
}
void display()
{
System.out.print(roll + " " + name);
}
}
class Result extends Exam
{
int marks;
Result()
{}
Result(int a, String s, int m)
{
super(a,s);
marks=m;
}
void display()
{
super.display();
```

```
System.out.println(" " +marks );
}
staticvoid sort(Result a[], int n)
{
Result temp = new Result();
for(int i=0;i<n;i++)
{
for(int j=(i+1);j<n;j++ )
{
if(a[i].marks<a[j].marks)
{
temp = a[i];
a[i]= a[j];
a[j]=temp;
}
}
}
}
}

public class Student {
```

```
public static void main(String[] args) {
Result a[] = new Result[5];

a[0] = new Result(2,"Arshid", 200);
a[1] = new Result(3,"Mehvish", 210);
a[2] = new Result(4,"Mushtaq", 100);
a[3] = new Result(5,"Feroz", 150);
a[4] = new Result(6,"Sadaf", 120);
System.out.println("Before Sorting ");
for(int i=0;i<5;i++)
a[i].display();
Result.sort(a, 5);
System.out.println("After Sorting based on Marks");
for(int i=0;i<5;i++)
a[i].display();
}
}
```

We will get the following output:

Before Sorting

2 Arshid 200

3 Mehvish 210

4 Mushtaq 100

5 Feroz 150

6 Sadaf 120

After Sorting based on Marks

3 Mehvish 210

2 Arshid 200

5 Feroz 150

6 Sadaf 120

4 Mushtaq 100

In the following Java program, the multilevel inheritance is implemented. The Father class is created which is inherited by the Student1 class and it is further inherited by the Result1 class:

```
package book;

class Father

{

String f_name;

Father()

{
```

```
}
Father(String nm)
{
f_name=nm;
}
}
class Student1 extends Father{
protected int roll,marks;
private String name;
Student1()
{
}
Student1(String fnm,int a,int b,String nm)
{
super(fnm);
roll=a;
marks=b;
name=nm;
}
void display()
{
```

```
System.out.println("Fathers Name =" + f_name+"\nRoll Number " + roll+"
\nName "+ name +"\nMarks " + marks);

}

}

public class Result1 extends Student1

{

private int sp_marks;

Result1()

{}

Result1(String fnm,int a,int b,String s,int m)

{

super(fnm,a,b,s);

sp_marks=m;

}

void display()

{

super.display();

System.out.println("Total Marks Are " + get_Total());

}

int get_Total()

{
```

```
return(sp_marks+marks);
}
public static void main(String arg[])
{
Result1 s1 =new Result1("Gh. Mohammad ",3,400,"Sajad",300);
s1.display();
}
}
```

We will get the following output:

Fathers Name =Gh. Mohammad

Roll Number 3

Name Sajadtt

Marks 400

Total Marks Are 700

Method overriding

If we have a method in a base class and same method name is redefined in the derived class the concept is called as method overriding. In a class ladder, when a method in a subclass has the identical name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. When an overridden method is called from inside a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be concealed. For example, the below segment of program overrides the show method in the base class:

```
class BaseClass

{

// Body of the class

void show( )

{

// Body of show ( ) method

}

}

class DerivedClass extends BaseClass

{

// Body of the class

void show( )
```

```
{

// Body of show ( ) method

}

}
```

Now, if we create an object of DerivedClass, it will have two show() methods. The show() method in its own class will be called. In order to call the show() method of the BaseClass, the super keyword is used. The following example illustrates the use of method overriding.

In the following Java program we have a base class and a derived class. The display method is defined in both the classes. If the object of derived class is used to access the display function it will go to its own display defined in its own class. Now if we want to call the display of the base class we have to use the super.display(); to call the display of base class:

```
package book;

class Student1 {


protected int roll,marks;

private String name;

Student1()

{

}
```

```
Student1(int a,int b,String nm)

{

roll=a;

marks=b;

name=nm;

}

void display()

{

System.out.println("Roll Number " + roll+" \nName "+ name +"\nMarks " +
marks);

}

}

public class Result1 extends Student1

{

private int sp_marks;

Result1()

{}

Result1(int a,int b,String s,int m)

{

super(a,b,s);

sp_marks=m;
```

```
}
void display()
{
super.display();
System.out.println("Total Marks Are " + get_Total());
}
int get_Total()
{
return(sp_marks+marks);
}
public static void main(String arg[])
{
Result1 s1 =new Result1(3,400,"Feroz",300);
s1.display();
}
}
```

We will get the following output:

Roll Number 3

Name Feroz

Marks 400

Total Marks Are 700

Understanding the super, final, and abstract keywords

Inheritance in Java makes use of some keywords which are frequently used. In order to properly implement the reusability concepts in Java we must understand these concepts properly. Let us understand these keywords one by one:

The super keyword

The super keyword in Java is used for two purposes as:

To invoke the constructor of a base class. By default the implicit or default constructor of base class is invoked by the derived class. If we want to change the sequence that is to invoke the constructor with arguments, then we must use the super(parameters) as the first line of code in the derived class constructor.

In case of method overriding, if we want to call the method in the base class then we must use the super.methodname() to call the method of the base class.

In the following example we have used multilevel inheritance to find out the sequence of constructors if the super keyword is not used. In this program Java implicitly makes use of super to invoke the default constructor of base class. If we can look at the output it first goes to the base class constructor and then to the derived class constructors:

```
//MultiLevel Inheritance

class A

{

int da;

A()
```

```
{
System.out.println("Constructor of A Class");
}
A(int a)
{
da=a;
}
}
class B extends A
{
B()
{
System.out.println("Constructor of B Class");
}
}
class C extends B
{
C()
{
System.out.println("Constructor of C Class");
}
```

```
}

public class Super_Demo {


public static void main(String[] args) {

C obj=new C();

}

}
```

We will get the following output:

Constructor of A Class

Constructor of B Class

Constructor of C Class

In this Java program which is the extension of previous program, we have used the super keyword to change the sequence of constructors, we used the super along with the parameters to call a parameterized constructor of base class:

```
package book;

class A

{

int da;
```

```
A()
{
System.out.println("Constructor of A Class");
}
A(int a)
{
da=a;
System.out.println("Parameterized Constructor of A Class");
}
}
class B extends A
{
int db;
B()
{
System.out.println("Constructor of B Class");
}
B(int a,int b)
{
super(a);
db=b;
```

```
System.out.println("Parameterized Constructor of B Class");
}
}
class C extends B
{
int dc;
C()
{
System.out.println("Constructor of C Class");
}
C(int a,int b,int c)
{
super(a,b);
dc=c;
System.out.println("Parameterized Constructor of C Class");
}
}
public class Super_Demo {
public static void main(String[] args) {
C obj=new C(2,3,4);
}
```

```
}
```

We will get the following output:

Parameterized Constructor of A Class

Parameterized Constructor of B Class

Parameterized Constructor of C Class

In this example program we have used method overriding by creating three show methods in all the three classes. Now to reach to the show methods of base class we must use the super.show() as is done in the program:

```
package book;

class A

{

int da;

A()

{

System.out.println(" Default Constructor of A Class");

}

A(int a)

{
```

```
da=a;
System.out.println("Parameterized Constructor of A Class");
}
void show()
{
System.out.println("Data of A =" + da);
}
}
class B extends A
{ int db;
B()
{
System.out.println(" Default Constructor of B Class");
}
B(int a,int b)
{
super(a);
db=b;
System.out.println("Parameterized Constructor of B Class");
}
void show()
```

```
{
super.show();
System.out.println("Data of B =" + db);
}
}
class C extends B
{
int dc;
C()
{
System.out.println("Constructor of C Class");
}
C(int a,int b,int c)
{
super(a,b);
dc=c;
System.out.println("Parameterized Constructor of C Class");
}
C(int a)
{
//super();
```

```
dc=a;
System.out.println("Parameterized Constructor of C Class");
}
void show()
{
super.show();
System.out.println("Data of C =" + dc);
}
}
public class Super_Demo {
public static void main(String[] args) {
C obj=new C(2,3,4);
C obj2=new C(2);
obj.show();
}
}
```

We will get the following output:

Parameterized Constructor of A Class

Parameterized Constructor of B Class

Parameterized Constructor of C Class

Default Constructor of A Class

Default Constructor of B Class

Parameterized Constructor of C Class

Data of A =2

Data of B =3

Data of C =4

The final keyword

The final keyword is used in the following scenarios:

To declare a class as final: If the class is declared as final it cannot be inherited. The syntax for declaring class as final is as under:

```
final ClassName
{
// Body of the class
}
```

To prevent method overriding: As we know method overriding is the concept of having a method in the base class and redefining the method with the same name in the derived class. If we don't want that the derived class should create the same method name, we can create a method in the base class as final. This will prevent the derived class to override that method. The syntax for using final method is:

```
final methodName()
{
}
```

Other important feature of final keyword is that we can create constants. If we use final with variables there values cannot be changed. For example the value of data cannot be changed as it has been declared as final.

```
final int data=100;
```

Let's understand the use of final with the following examples:

In the following example we created a final class MyClass and tried to inherit it but as can be seen in the error message that class B cannot inherit the final class:

```
1  package book;
2  final class MyClass
3  {
4
5  }
6  class B extends MyClass
7  {
8
9  }
10 public class Fin_____ {
11
12     public static void main(String[] args) {
13         // TODO Auto-generated method stub
14
15     }
16
17 }
18
```

The type B cannot subclass the final class MyClass

1 quick fix available:

Remove 'final' modifier of 'MyClass'

Press 'F2' for focus

Figure 3.7: The final class cannot be inherited final class cannot be inherited

In this Java program the method is created as final method and Java is not allowing us to override the final method in the base class:

```
1  package book;
2  class MyClass
3  {
4  final void show()
5  {
6      System.out.println("Final Method Can't get Overriden" );
7  }
8  }
9  class B extends MyClass
10 {
11     void show()
12     {
13
14     }
15 }
16 public class Final_Demo {
17
18     public static void main(String[] args) {
19         // TODO Auto-generated method stub
20
21     }
22
23 }
```

Cannot override the final method from MyClass

1 quick fix available:

Remove 'final' modifier of 'MyClass.show()'

Press 'F2' for focus

Figure 3.8: The final method cannot be overridden

In this Java program we tried to change the contents of a final data member:

```
1  package book;
2  class MyClass
3  {
4  final void show()
5  {
6      System.out.println("Final Method Can't get Overriden" );
7  }}
8  class B extends MyClass
9  {
10 /*  void show()
11     {
12         // Not Able to Override
13     }
14     */
15 }
16 public class Final_Demo {
17
18     public static void main(String[] args) {
19         final int data=100;
20         data=data+2;
21     }
22 }
23
```

The final local variable data cannot be assigned. It must be blank and not using a compound assignment

1 quick fix available:

Remove 'final' modifier of 'data'

Press 'F2' for focus

Figure 3.9: The final data example

The abstract keyword

The abstract keyword is used with the method or with the classes. If it is used while defining a method, it indicates that this method must be redefined in the derived class or in other words it must be overridden. This means that abstract keyword is opposite to final keyword. If the abstract keyword is used with class name, it indicates that we must inherit that class and we cannot directly create the objects of that class. We cannot use the abstract with the constructors and the static methods. The below examples illustrates the use of abstract.

In the following Java program an abstract super class with abstract methods are created. The actual definition of these methods is implemented in the derived class. It is pertinent to mention here that the abstract methods have no body and all abstract methods must be defined in the derived class:

```
package book;


abstract class MyClass

{

abstract void show();

abstract void display();

}

class B extends MyClass

{
```

```
void show() {
System.out.println("Actual Implementation of show method");
}
void display() {
System.out.println("Actual Implementation of display method");
}
}
public class Abstract_Demo {
public static void main(String[] args) {
B obj = new B();
obj.show();
obj.display();
}
}
```

We will get the following output:

Actual Implementation of show method

Actual Implementation of display method

Assignments

What are the various forms of inheritance in Java? Write a Java program to implement the multilevel inheritance?

How does the final keyword affect the class and a method and how it is different from the abstract class and abstract method?

What is the importance of a constructor and how does the super keyword affect the sequence of constructors?

Write a Java program that gets the employee details from the command line and create an array of objects of employee class and sort those objects based on the salary?

CHAPTER 4

Packages and Interfaces in Java

Structure

Introduction

Creating our own packages

Creating a package within a package

Access control with packages

Understanding Java inbuiltpackages

Interfaces

Understanding the implementation of interfaces

Extending the interfaces

Understanding the built in interfaces.

Assignments

Objective

The inbuilt packages of Java are explained in this chapter. The reader will be able to create its own packages and will be able to use the built-in packages of Java.

Introduction

Java programming is based on object-oriented methodology and the core of it is the classes and objects. If we put similar classes together to define same type of functionality the concept is called as package in Java. For example one of the built in package in Java is theJava.io package which includes all the classes related to input and output and if we talk about the event package that holds all those classes which are required for event handling. Therefore, a package also called as a namespace, groups the classes together to be used for a similar type of functionality. The main advantage of packages is the reusability of the code that we discussed in previous chapters in terms of inheritancethat can be enhanced if we want to use the classes from other programs that can be done by making use of a package and importing that package in our own program.

The Java has built in packages like lang, awt, util,io,applet, and more. These packages have numerous classes which can be easily reused in our Java program for example if we are in need of a Button class, we need not to create that class from the scratch instead we can directly import that class from the awt package of Java. So the advantages of packages in Java are many but some of the advantages are as under:

The classes in built-in packages of Java or created by other programmers can be easily reused through packages.

We can have same class names in two different packages. The classes can be called by fully qualified names to be discussed in upcoming sections.

Data hiding can be achieved through packages.

Packages allow the programmer to separate design from coding.

Creating our own packages

In order to understand the built in packages in Java, we should be able to create our own package and put N number of classes in this package. We will create a package nielit and put three classes namely MCA, IT and BCA within this package. The steps that we need to follow are as under:

Create a subdirectory with the same name as nielit. If we are using the Command Prompt we can do that using the command:

md nielit

In Eclipse we should right click on the package that we are in and click on New Package as under:

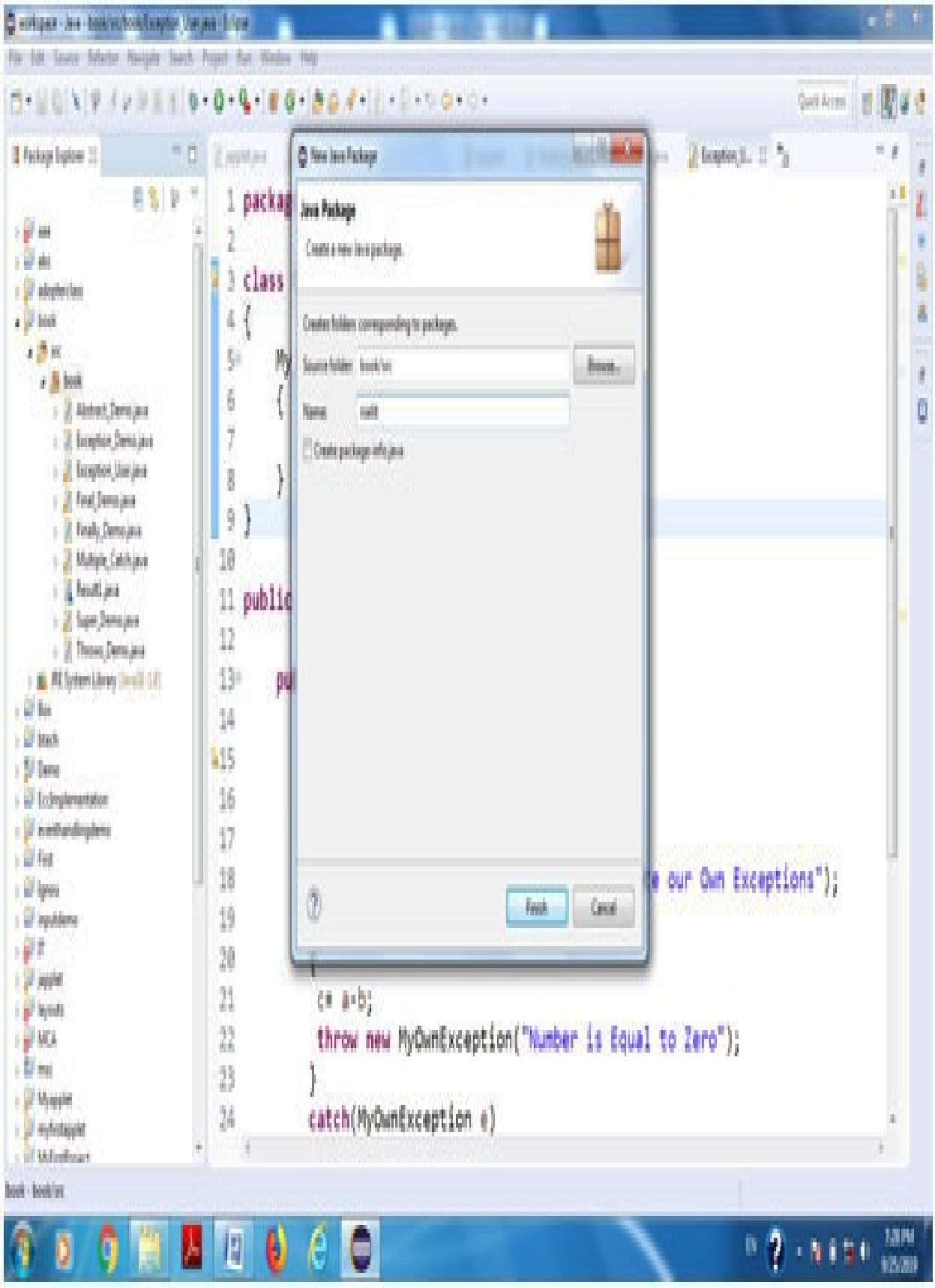

New Java Package
Java Package
Create a new Java package.
Create folders corresponding to packages.
Source folder:
Browse...
Name:
Create package-info.java
Finish
Cancel
e our Own Exceptions");
ce a=b;
throw new MyOwnException("Number is Equal to Zero");
}
catch(MyOwnException e)

Figure 4.1: Creation of package

Now after typing the name of the package as nielit, we will right click on the nielit package to create the classMCA of our nielit package:

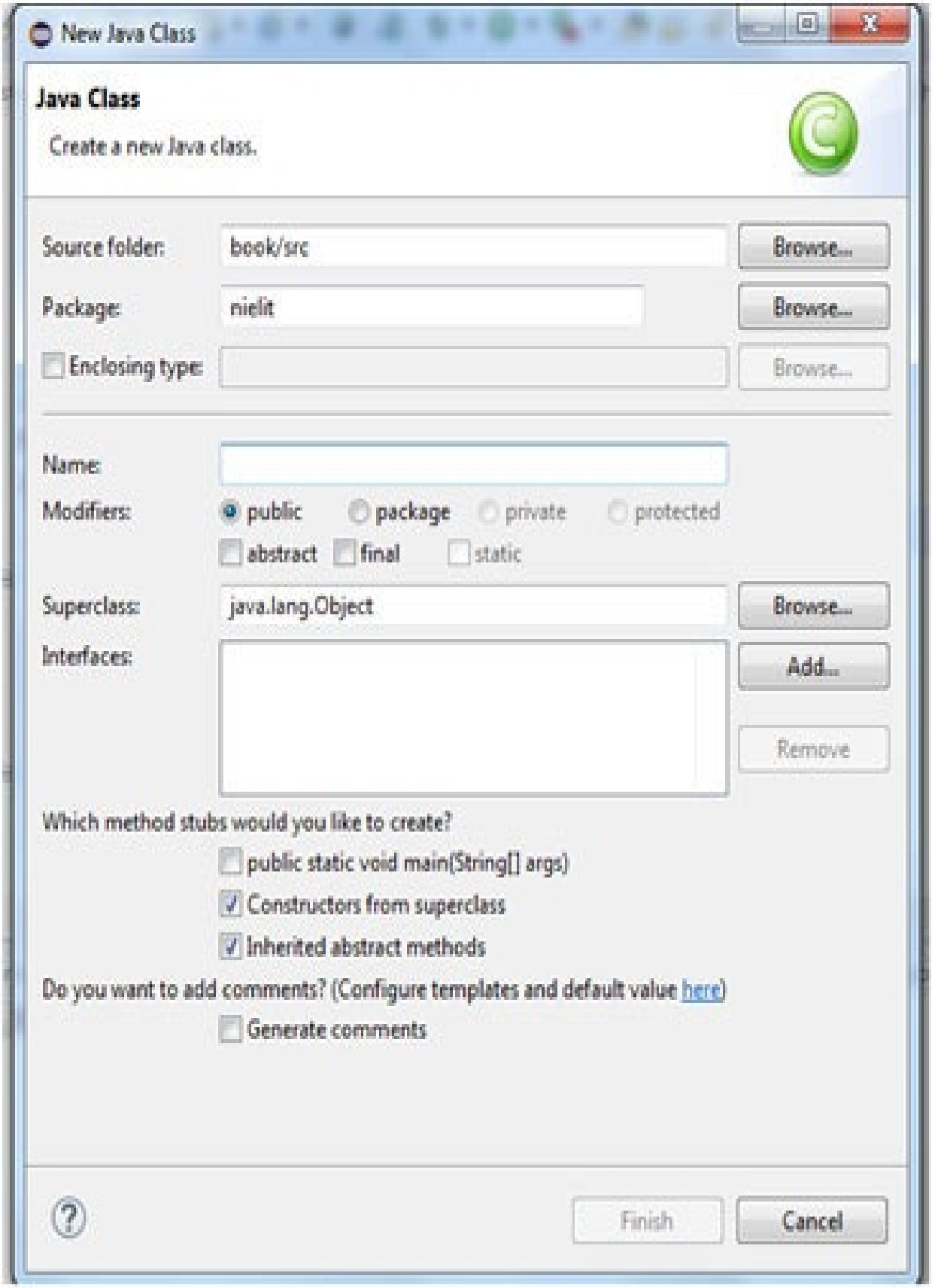

New Java Class
Java Class
Create a new Java class.
Source folder:
book/src
Browse...
Package:
nielit
Browse...
Enclosing type:
Browse...
Name:
Modifiers:
public
package
private
protected
abstract
final
static
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

Figure 4.2: Creation of a class within a package

The class definition is as under. The first line of this class starts with the name of the package as this class belongs to nielit package so the first line is package nielit:

```
// Class MCA

package nielit;

public class MCA {

public MCA()

{

System.out.println("Constructor of MCA");

}

public void showMCA()

{

System.out.println("MCA Class within nielit package");

}

}
```

In the same way we should include IT and BCA classes:

```
// IT Class
package nielit;
public class IT {
public IT()
{
System.out.println("Constructor of IT");
}
public void showIT()
{
System.out.println("IT Class within nielit package");
}
}
// BCA Class
package nielit;
public class BCA {
public BCA()
{
System.out.println("Constructor of BCA");
}
public void showBCA()
{
```

```
System.out.println("BCA Class within nielit package");

}

}
```

After creating all the three classes within the nielit package, the next step is to compile these classes. We have not put the main methods within these classes as we want to reuse those classes in some other program.

Now after successfully compiling the classes we will create a class in the parent package to import all the classes and reuse them. The new class is created as under:

```
package book;

import nielit.*;

public class Package_Demo {

public static void main(String[] args) {

MCA m1 =new MCA();

IT m2= new IT();

BCA m3 = new BCA();

m1.showMCA();

m2.showIT();

m3.showBCA();

}
```

}

In order to import all the classes we used the import statement as:

import nielit. *;

The (*) indicates all the classes from this package. If suppose we want to import only the MCA class, we can use import nielit.MCA; so the output of the above program is as under:

We will get the following output:

Constructor of MCA

Constructor of IT

Constructor of BCA

MCA Class within nielit package

IT Class within nielit package

BCA Class within nielit package.

Creating a package within a package

Our next objective is to create a package within a package. The hierarchical representation is as under:

nielit package

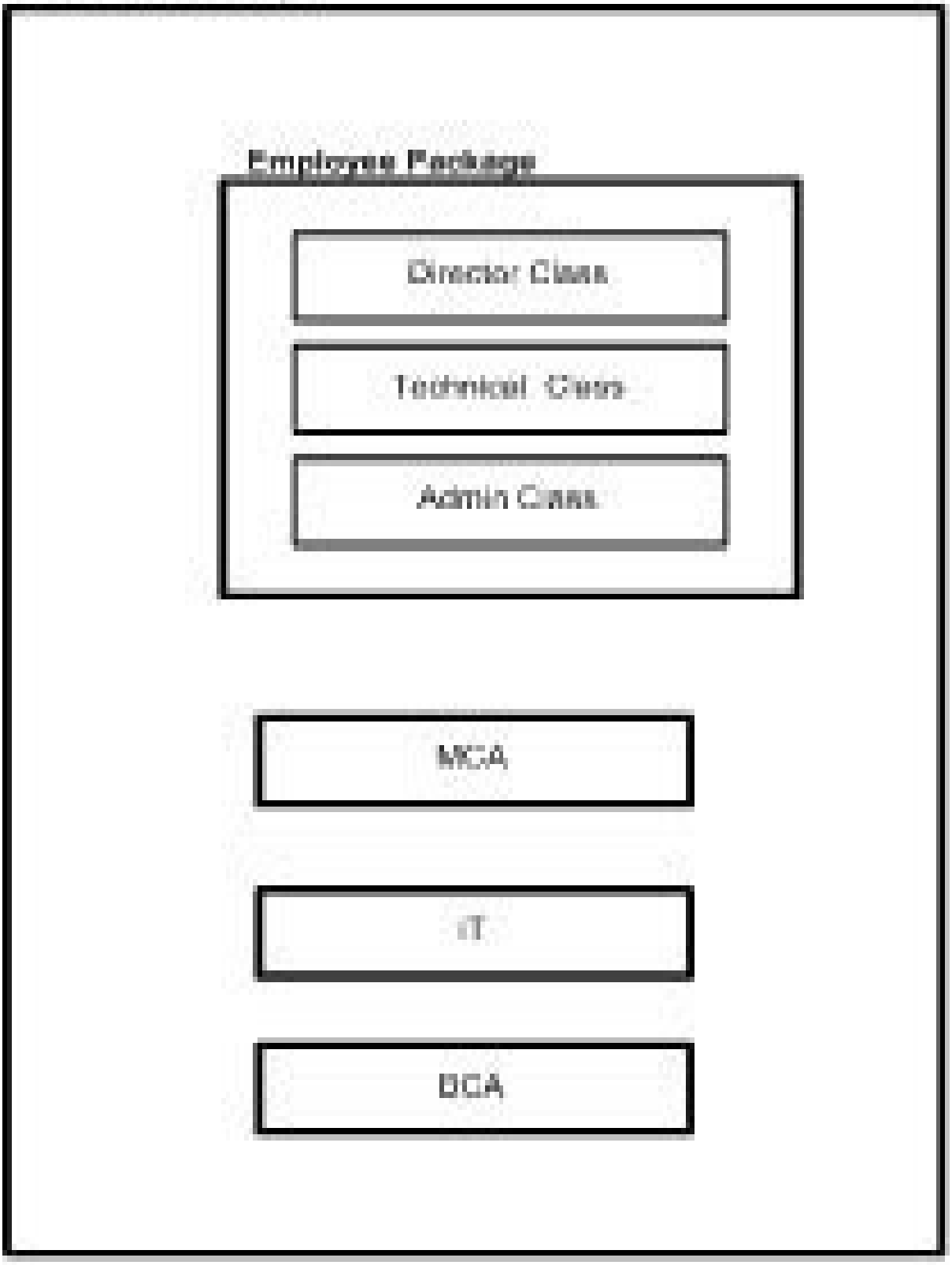


Figure 4.3: Package within a package

Now from the above diagram, we have a package nielit and a sub package Employee. The Employee package has three classes namely Director, Technical, and Admin class. As we have already implemented the nielit package in the previous section, we can move forward by adding the subpackage to the nielit package by incorporating the following steps:

Create a subpackage under nielit package by right clicking on nielit package and click on New Package. Name this subpackage as employee.

Create the following classes and write the first line of code as package nielit.employee as all these three classes belong to the nielit.employee package:

```
// Director Class

package nielit.employee;

public class Director {

public Director()

{

System.out.println("Constructor of Director");

}

public void showDir()
```

```
{
System.out.println("Director Class within nielit.employee package");
}
}
// Technical Class
package nielit.employee;
public class Technical {
public Technical()
{
System.out.println("Constructor of Technical");
}
public void showTechnical()
{
System.out.println("Technical Class within nielit package");
}
}
// Admin Class
package nielit.employee;
public class Admin {
public Admin()
{
```

```
System.out.println("Constructor of Admin");

}

public void showAdmin()

{

System.out.println("Admin Class within nielit package");

}

}
```

Now after successfully compiling all the three classes, it is time to reuse them. The following statements are used to import all the classes.

import nielit.employee.*; and import nielit.*;

It is mandatory to use the fully qualified name to import all the classes within the nielit package and its subpackage:

```
package book;

import nielit.employee.*;

import nielit.*;

public class Package_Demo {

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

```
MCA m1 =new MCA();

IT m2= new IT();

BCA m3 = new BCA();

Technical t = new Technical();

m1.showMCA();

m2.showIT();

m3.showBCA();

t.showTechnical();

}

}
```

We will get the following output:

Constructor of MCA

Constructor of IT

Constructor of BCA

Constructor of Technical

MCA Class within nielit package

IT Class within nielit package

BCA Class within nielit package

Technical Class within nielit package

From the above discussion the following points must be taken into consideration:

The package to which a class belongs is written as a first line of the code and the syntax for that is

package package_name;

The package name always starts with a small letter.

The import statement can take the following forms:

import package_name.*; // To import all the classes

OR

import package_name.subpackage.*; // To import all the classes from sub package

OR

import package_name.ClassName; // To import only a single class from the package.

Access control with packages

In the earlier chapters, we studied regarding different aspects of Java's access control mechanism and its access specifiers. For example, wepreviously understood that access to a private members of a class is approved just to other members of that class. Packages attach additionalaspect to access control. As we will see, Java makes available many levels of security to permit fine-grained control above the visibility of variables and methods within classes, subclasses, and packages. Classes and packages are in togetherways of encapsulating and containing the name space and scope of variables and methods. Packages act as containers for classes and other subordinate packages. Classes act as containers for data and code. The class is Java'sminimum unit of abstraction. Because of the relationship between classes and packages, Java addresses four categories of visibility for class members:

Subclasses in the same package

Non-subclasses in the same package

Subclasses in different packages

Classes that are neither in the same package nor subclasses

The three access specifiers, private, public, and protected, provide a variety of ways to produce the many levels of access required by these categories. The

table below sums up the interactions:

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Table 4.1: Member Access with packages

In order to understand the visibility of class members let us take the following example:

```
// Class BCA within the nielit package

package nielit;

public class BCA {

public int pub_data;

private int pri_data;

protected int pro_data;

public BCA()

{

System.out.println("Constructor of BCA");

}

public void showBCA()

{

System.out.println("BCA Class within nielit package");

}

}
```

The three data members in the above class are declared as public, private and protected. If we want to access these members from within the same package as done in the following class. We will get the error for the private data as it is only accessed from within the same class in which it is declared:

```
package nielit;

public class IT extends BCA

{

public IT()

{

System.out.println("Constructor of IT");

}

public void showData()

{

System.out.println("Public Data =" + pub_data);

System.out.println("Private Data =" + pri_data); // Error as Private Data is not Accessible

System.out.println("Protected Data =" + pro_data);

}

public void showIT()

{

System.out.println("IT Class within nielit package");
```

```
}

}
```

Now suppose we will not extend the IT class from the BCA class then the private data will only generate the error as under:

```
package nielit;

public class IT

{

public IT()

{

System.out.println("Constructor of IT");

}

public void showData()

{

BCA obj = new BCA();

System.out.println("Public Data =" + obj.pub_data);

System.out.println("Private Data =" + obj.pri_data); // Error as Private Data is not Accessible

System.out.println("Protected Data =" + obj.pro_data);

}

public void showIT()
```

```
{

System.out.println("IT Class within nielit package");

}

}
```

Now, we want to access the same data from other package, the private and protected data will generate the errors as under:

```
package book;

import nielit.BCA;

public class IT

{

public IT()

{

System.out.println("Contructor of IT");

}

public void showData()

{

BCA obj = new BCA();

System.out.println("Public Data =" + obj.pub_data);

System.out.println("Private Data =" + obj.pri_data); // Error as Private Data is not Accessible
```

```
System.out.println("Protected Data =" + obj.pro_data); // Error

}

public void showIT()

{

System.out.println("IT Class within nielit package");

}

}
```

If we extend the above class from the BCA class, then the protected data will not generate an error and only the private data will generate the error as under:

```
package book;

import nielit.BCA;

public class IT extends BCA

{

public IT()

{

System.out.println("Constructor of IT");

}

public void showData()

{
```

```
System.out.println("Public Data =" + pub_data); // OK

System.out.println("Private Data =" + pri_data); // Error as Private Data is not Accessible

System.out.println("Protected Data =" + pro_data); // OK

}

public void showIT()

{

System.out.println("IT Class within nielit package");

}

}
```

From the above illustrations, it is clear that the public data is accessible from anywhere while as private members are only visible within the class in which they are defined. The protected members are visible from within the same package and from outside the package if the class is a subclass.

Understanding Java inbuilt packages

Most of the functionality in Java is implemented through classes stored in different Java API packages. The classes are grouped together based on their functionalities and are provided to a programmer in the form of a package. Some of the frequently used Java packages are as under:

Package Name	Classes
java.lang	It includes the language support classes suchas. Integer, String, T
java.util	It includes vast variety of utility classes like Date, Random, and
java.awt	In order to create a GUI type of an application, this package is ir
java.io	To deal with input / output in Java, the io package is required. T
java.sql	This sql package is used for the database connectivity. It include
java.event	Event Handling is implemented through java.event. This packag
java.applet	It includes the Applet class used to create the user defined apple

Table 4.2: Inbuilt packages in Java

java.lang package

The java.lang is the default package in Java. If we want to use any of its class, we need not to import it as it is imported automatically to our Java program. The following table lists some of the classes available in java.lang:

Boolean	InheritableThreadLocal	Runtime	System
Byte	Integer	RuntimePermission	Thread
Character	Long	SecurityManager	ThreadGroup
Class	Math	Shor t	ThreadLocal
ClassLoader	Number	StackTraceElement	Throwable
Compiler	Object	StrictMath	Void
Double	Package	String	Exception
Enum	Process	StringBuffer StringBuilder	
Float	ProcessBuilder		

Table 4.3: Classes in java.lang

The interfaces in java.lang are listed as under:

Appendable	Comparable	Runnable
CharSequence	Iterable	
Cloneable	Readable	

Table 4.4: interfaces in java.lang

The following Java program uses one of the important class of java.lang package. The Runtime class is used to get the details about the memory availability at runtime. The different methods of Runtime are used in this program. To execute other programs from a Java program, the Runtime class is used. In this program we have executed the calculator by calling the exec method of the Runtime. As this package is imported automatically, the import statement is not needed:

```
package book;

public class LangExample {

public static void main(String args[]) {

Runtime r = Runtime.getRuntime();

Process p = null;

long first, second;

String data[] = new String[1000];

System.out.println("Memory =: " +r.totalMemory());

first = r.freeMemory();

System.out.println("Free Memory =: " + first); r.gc();

first = r.freeMemory();

System.out.println("After Garbage Collection: "+ first);

for(int i=0; i<1000; i++)
```

```
data[i] = new String("i");
second = r.freeMemory();
System.out.println("After allocation: " +second);
System.out.println("Memory used by String Array: " +(first-second));
for(int i=0; i<1000; i++) data[i] = null;
r.gc(); // request garbage collection
second = r.freeMemory();
System.out.println("Free memory after collecting discarded Strings: " + second);
System.out.println(" Using Runtime to open a Calculator ");
try {
p = r.exec("calc");
} catch (Exception e)
{ System.out.println("Error Opening the Calculator.");
}
}
}
```

We will get the following output:

Memory =: 192937984

Free Memory =: 190924680

After Garbage Collection: 192624288

After allocation: 191617608

Memory used by String Array: 1006680

Free memory after collecting discarded Strings: 192626168

Using Runtime to open a Calculator

The following program implements Cloneable and defines the method cloneTest(), which calls super.clone() in Object. The cloning of object obj1 is done through this interface which is available in java.lang package:

```
package book;

class CloneExample implements Cloneable {

String data_1;

double data_2;

CloneExample cloneTest() {

try {

return (CloneExample)

super.clone();

} catch(CloneNotSupportedException e)

{

System.out.println("Cloning not allowed.");

return this;
```

```
}
}
}

class LangDemoExample {
public static void main(String args[]) {
CloneExample obj1 = new CloneExample();
CloneExample obj2;
obj1.data_1 = "Hello Java";
obj1.data_2 = 4.7;
obj2 = obj1.cloneTest(); // clone obj1
System.out.println("obj1: " + obj1.data_1 + " " + obj1.data_2);
System.out.println("obj2: " + obj2.data_1 + " " + obj2.data_2);
}
}
```

We will get the following output:

obj1: Hello Java 4.7

obj2: Hello Java 4.7

java.util

The following table lists the classes available in java.util package. Some of the most useful classes are: Random – Used to generate the random numbers, Scanner- Used to get the input from the user, Calendar, Date, Vector and may others.

AbstractCollection	EventObject	Random
AbstractList	FormattableFlags	ResourceBundle
AbstractMap	Formatter	Scanner
AbstractQueue	GregorianCalendar	ServiceLoader
AbstractSequentialList	HashMap	SimpleTimeZone
AbstractSet	HashSet	Stack
ArrayDeque	Hashtable	StringTokenizer
ArrayList	IdentityHashMap	Timer
Arrays	LinkedHashMap	TimerTask
BitSet	LinkedHashSet	TimeZone
Calendar	LinkedList	TreeMap
Collections	ListResourceBundle	TreeSet
Currency	Locale	UUID
Date	Observable	Vector
Dictionary	PriorityQueue	WeakHashMap
EnumMap	Properties	
EnumSet	PropertyPermission	

EventListenerProxy	PropertyResourceBundle	

Table 4.5: Classes available in java. util

The interfaces available in java.util are as under:

Collection	List	Queue
Comparator	ListIterator	RandomAccess
Deque	Map	Set
Enumeration	Map.Entry	SortedMap
EventListener	NavigableMap	SortedSet
Formattable	NavigableSet	
Iterator	Observer	

Table 4.6: Interfaces in java.util

In the belowJava program the Arrays class of java.util package is explored. The Arrays class is useful in dealing with arrays. The methods available in Arrays class like sort and binarySearch are used in this program:

```
package book;

import java.util.*;

public class UtilDemo {

public static void main(String args[]) {

int array[] = new int[10];

for(int i = 0; i < 10; i++)

array[i] = 5 * i;

System.out.print("Original contents: ");

display(array);

Arrays.sort(array);

System.out.print("Sorted: ");

display(array);

Arrays.fill(array, 2, 4, 3);

System.out.print("After fill(): ");

display(array);
```

```
Arrays.sort(array);
System.out.print("After sorting again: ");
display(array);
System.out.print("The value 25 is at location ");
int index = Arrays.binarySearch(array, 25);
System.out.println(index);


}
static void display(int array[]) {
for(int i: array)
System.out.print(i + " ");
System.out.println();
}
}
```

We will get the following output:

Original contents: 0 5 10 15 20 25 30 35 40 45

Sorted: 0 5 10 15 20 25 30 35 40 45

After fill(): 0 5 3 3 20 25 30 35 40 45

After sorting again: 0 3 3 5 20 25 30 35 40 45

The value 25 is at location 5

In this Java program, one of the most important utility class called as Calendar is used:

```
package book;

import java.util.Calendar;

public class UtilExample {

public static void main(String args[]) {

String months[] = {

"January", "February", "March", "April",

"May", "June", "July", "August",

"September", "October", "November", "December"};


Calendar calendar = Calendar.getInstance();

System.out.print("Date: ");

System.out.print(months[calendar.get(Calendar.MONTH)]);

System.out.print(" " + calendar.get(Calendar.DATE) + " ");

System.out.println(calendar.get(Calendar.YEAR));

System.out.print("Time: ");

System.out.print(calendar.get(Calendar.HOUR) + ":");
```

```
System.out.print(calendar.get(Calendar.MINUTE) + ":");
System.out.println(calendar.get(Calendar.SECOND));
calendar.set(Calendar.HOUR, 10);
calendar.set(Calendar.MINUTE, 29);
calendar.set(Calendar.SECOND, 22);
System.out.print("Updated time: ");
System.out.print(calendar.get(Calendar.HOUR) + ":");
System.out.print(calendar.get(Calendar.MINUTE) + ":");
System.out.println(calendar.get(Calendar.SECOND));
}
}
```

We will get the following output:

Date: October 22 2019

Time: 9:12:58

Updated time: 10:29:22

java.awt

The AWT classes are enclosed in the java.awt package. It is one of Java's biggest packages. Some of the classes of awt package are as under:

Class	Description
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five com
Button	Creates a push button object.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index ca
Checkbox	Creates a check box object.
CheckboxGroup	Creates a group of check box objects.
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five com
Button	Creates a push button object.
Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate index ca
Checkbox	Creates a check box object.
CheckboxGroup	Creates a group of check box objects.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.

Color	Manages colors in a portable, platform-independent fash
Component	An abstract superclass for various AWT components.
Container	A subclass of Component that can hold other component
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is store
Event	Encapsulates events.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions compon
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font. This
Frame	Creates a standard window that has a title bar, resize cor
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fash
Component	An abstract superclass for various AWT components.
Container	A subclass of component that can hold other component
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is store
Event	Encapsulates events.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions compon
Font	Encapsulates a type font.

FontMetrics	Encapsulates various information related to a font. This
Frame	Creates a standard window that has a title bar, resize cor
Graphics	Encapsulates the graphics context. This context is used b
GraphicsDevice	Describes a graphics device such as a screen or printer.
GraphicsEnvironment	Describes the collection of available Font and GraphicsI
GridBagConstraints	Defines various constraints relating to the GridBagLayo
GridBagLayout	The grid bag layout manager. GridBag layout displays c
GridLayout	The grid layout manager. Grid layout displays componer
Image	Encapsulates graphical images.
Insets	Encapsulates the borders of a container.
Label	Creates a label that displays a string.
List	Creates a list from which the user can choose. Similar to
MediaTracker	Manages media objects.
Menu	Creates a pull-down menu.
MenuBar	Creates a menu bar.
MenuComponent	An abstract class implemented by various menu classes.
MenuItem	Creates a menu item.
MenuShortcut	Encapsulates a keyboard shortcut for a menu item.
Panel	The simplest concrete subclass of Container.
Point	Encapsulates a Cartesian coordinate pair, stored in Xand
Polygon	Encapsulates a polygon.
PopupMenu	Encapsulates a pop-up menu.
PrintJob	An abstract class that represents a print job.
Rectangle	Encapsulates a rectangle.
Robot	Supports automated testing of AWT-based applications.
Scrollbar	Creates a scroll bar object.

ScrollPane	A container that provides horizontal and/or vertical scrol
SystemColor	Contains the colors of GUI widgets such as windows, sc
TextArea	Creates a multiline edit object.
TextComponent	A superclass for TextArea and TextField.
TextField	Creates a single-line edit object.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar, and no ti

Table 4.7: Some of the classes in java.awt package

Now in order to understand the importance of built in packages the following programs describe the usage of these packages.

The following program makes use of several classes and all those classes are imported from different package like awt, event, swing, and many more. For a programmer, it was very difficult to write such a big program without the support of these built in packages.

```
// Program for designing the GUI game.

package swingsdemo;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.*;

import javax.swing.border.Border;

public class swingsdemo implements ActionListener {

static String Msg = new String("X");

static Button b1= new Button();

static Button b2= new Button();

static Button b3= new Button();
```

```
static Button b4= new Button();
static Button b5= new Button();
static Button b6= new Button();
static Button b7= new Button();
static Button b8= new Button();
static Button b9= new Button();
public static void main(String[] args) {
// TODO Auto-generated method stub
JFrame jf = new JFrame("My Project");
jf.setSize(500, 500);
Container c;
c= jf.getContentPane();
c.setLayout(null);
c.add(b1).setBounds(50, 100, 70, 70);
c.add(b2).setBounds(150, 100, 70, 70);
c.add(b3).setBounds(250, 100, 70, 70);
c.add(b4).setBounds(50, 200, 70, 70);
c.add(b5).setBounds(150, 200, 70, 70);
c.add(b6).setBounds(250, 200, 70, 70);
c.add(b7).setBounds(50, 300, 70, 70);
c.add(b8).setBounds(150, 300, 70, 70);
```

```
c.add(b9).setBounds(250, 300, 70, 70);
b1.addActionListener(new swingsdemo() );
b2.addActionListener(new swingsdemo());
b3.addActionListener(new swingsdemo());
b4.addActionListener(new swingsdemo());
b5.addActionListener(new swingsdemo());
b6.addActionListener(new swingsdemo());
b7.addActionListener(new swingsdemo());
b8.addActionListener(new swingsdemo());
b9.addActionListener(new swingsdemo());
jf.show();
}
@Override
public void actionPerformed(ActionEvent arg0) {
// TODO Auto-generated method stub
Button btn=(Button)arg0.getSource();
btn.setLabel(Msg);
btn.setEnabled(false);
CheckGame();
if (Msg.equalsIgnoreCase("X"))
Msg="0";
```

```
else
Msg="X";
}
private void CheckGame() {
// TODO Auto-generated method stub
if (b1.equals("X") && b5.equals("X") && b9.equals("X"))
{
b1.setLabel("Player 1 Won");
}
}
}
```

We will get the following output:

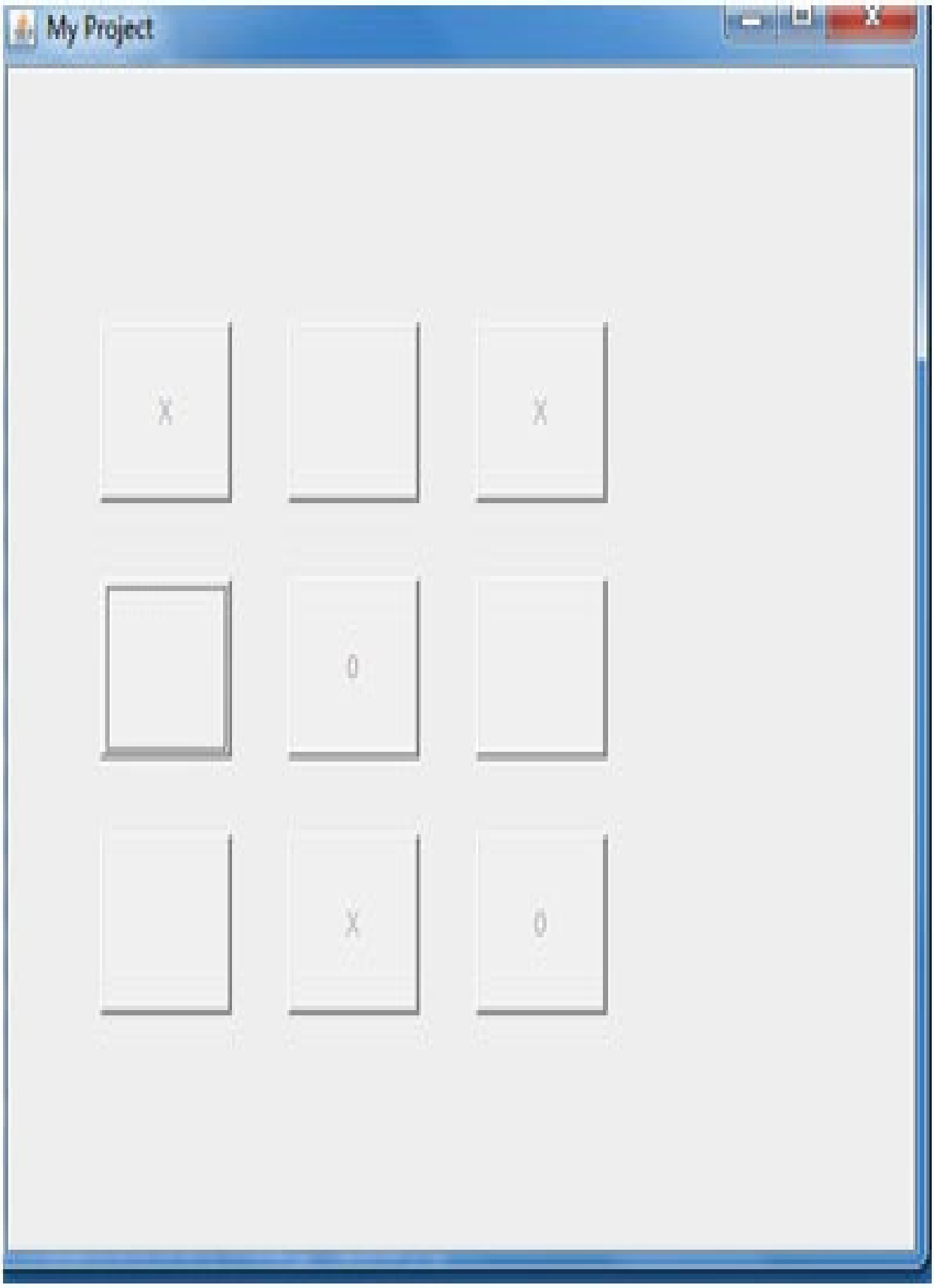

My Project
X
X
0
X
0

Figure 4.4

The following program shows the implementation of an Applet through java.applet package:

```
package myfirstapplet;

import java.applet.*;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.*;

import java.awt.event.*;

import java.net.URL;

import java.util.Locale;

import javax.accessibility.AccessibleContext;

public class abc extends Applet implements MouseListener

{

int x,y;

public void init()

{
```

```
Button b=new Button("Ok");
add(b);
b.addMouseListener(this);
addMouseListener(this);
}
public void paint(Graphics g)
{
g.drawString("*", x, y);
}
@Override
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
x= arg0.getX();
y= arg0.getY();
repaint();
}

@Override
public void mouseEntered(MouseEvent arg0) {
// TODO Auto-generated method stub
}
```

```
@Override
public void mouseExited(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mousePressed(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseReleased(MouseEvent arg0) {
// TODO Auto-generated method stub
}
}
```

We will get the following output:

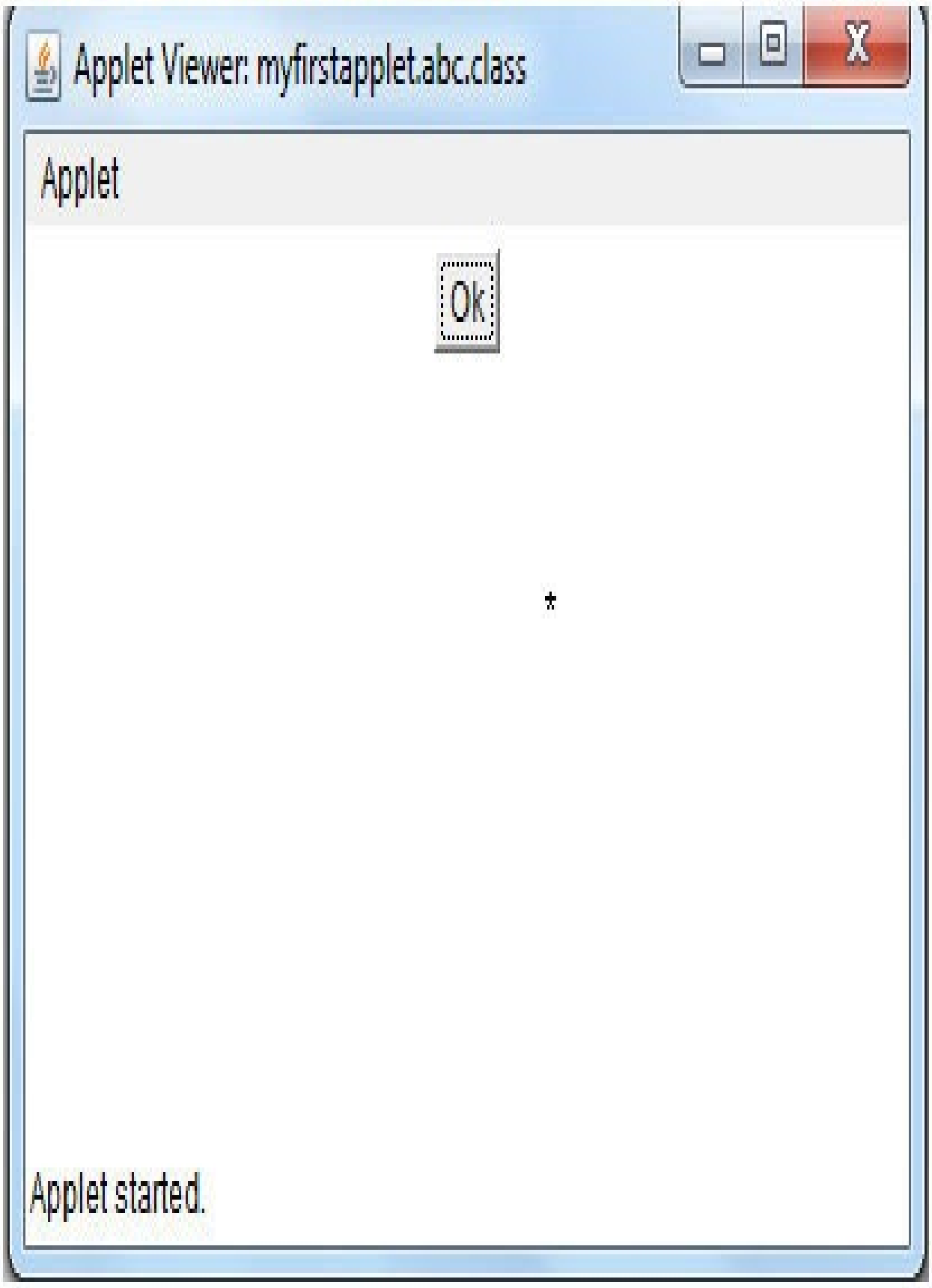

Applet Viewer: myfirstapplet.abc.class
Applet
Ok
Applet started.

Figure 4.5

Interfaces

The interfaces in Java are used to implement the multiple inheritances as we cannot extend two classes directly. So one has to be a class and one interface. But the interface includes only the abstract methods and final data so interfaces are in many ways different from classes as in interface we can have only final data that is the constant data while as in class we may have a constant or variable data. The other differences are quite obvious as class may have abstract or final method while as in interface only abstract methods without the definition is declared. The other major difference is that the interface only uses public while as a class can use public, private, and other access modifiers. If a class implements the interface, it is the responsibility of that class to implement all of its abstract methods. The general syntax of interface is as under:

```
interface InterfaceName

{

variable declaration;

method declaration

}
```

Interfaces

Figure 4.6: Interface

For example if we want to create an interface with name MyInterface having two abstract methods and one data member then the following declaration is required:

```
interface MyInterface

{

static final int data=100;

void show();

void display();

}
```

Now if the class MainClass wants to implement the above interface, it has to implement all the abstract methods of MyInterface and they must be declared as public as done under:

```
class MainClass implements MyInterface

{

public void show()

{

// Definition of show method
```

```
}

public void display()

{

}

}
```

The following examples will further illustrate the use of interface. In this Java program an interface is created and is implemented by a class named as Class1. The two abstract methods of MyInterface are implemented in Class1:

```
package book;

interface MyInterface

{

static final int data = 50;

void method1();

void method2();

}

class Class1 implements MyInterface

{

@Override

public void method1() {

System.out.println(" From Method 1");
```

```
}
@Override
public void method2() {
System.out.println(" From Method 2");
}
}
public class Interface_Demo
{
public static void main(String[] args) {
System.out.println("Interface Demo");
Class1 obj = new Class1();
obj.method1();
obj.method2();
}
}
```

We will get the following output:

Interface Demo

From Method 1

From Method 2

In the following Java program the interface SportsMarks is implemented and its methods are defined in the derived class:

```
class Exam

{

int roll;

String name;

Exam()

{}

Exam(int a, String s)

{

roll=a;

name=s;

}

void display()

{

System.out.print(roll + " " + name);

}

}

interface SportsMarks
```

```
{
static final int sp_marks=50;
void show_sports();
}
class Result extends Exam implements SportsMarks
{
int marks;
Result()
{}
Result(int a, String s, int m)
{
super(a,s);
marks=m + sp_marks;
}
void display()
{
super.display();
System.out.println(" " +marks );
}
staticvoid sort(Result a[], int n)
{
```

```
Result temp = new Result();
for(int i=0;i<n;i++)
{
for(int j=(i+1);j<n;j++)
{
if(a[i].marks<a[j].marks)
{
temp = a[i];
a[i]= a[j];
a[j]=temp;
}
}
}
}
@Override
public void show_sports() {
System.out.println("Sports Marks = " + sp_marks);
}
}
public class Student {
public static void main(String[] args) {
```

```
Result a[] = new Result[5];
a[0] = new Result(2,"Maheen", 200);
a[1] = new Result(3,"Hina", 210);
a[2] = new Result(4,"Mehak", 100);
a[3] = new Result(5,"Mareen", 150);
a[4] = new Result(6,"Sadaf", 120);
System.out.println("Before Sorting ");
for(int i=0;i<5;i++)
a[i].display();
Result.sort(a, 5);
System.out.println("After Sorting based on Marks and Sports marks");
for(int i=0;i<5;i++)
a[i].display();
}
}
```

We will get the following output:

Before Sorting

2 Maheen 250

3 Hina 260

4 Mehak 150

5 Mareen 200

6 Sadaf 170

After Sorting based on Marks and Sports marks

3 Hina 260

2 Maheen 250

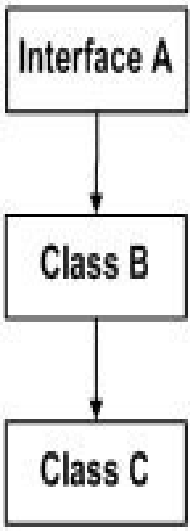
5 Mareen 200

6 Sadaf 170

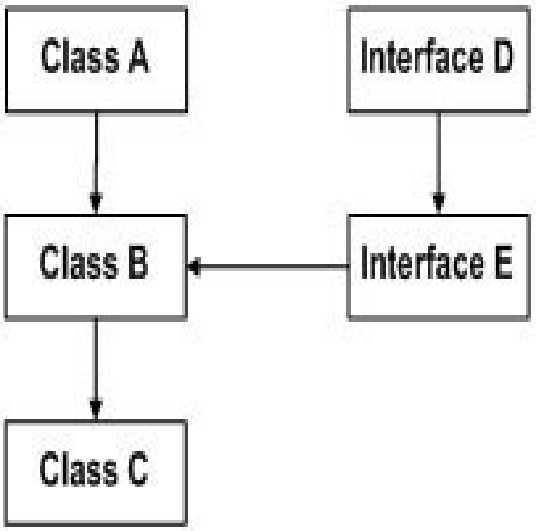
4 Mehak 150

Extending the interfaces

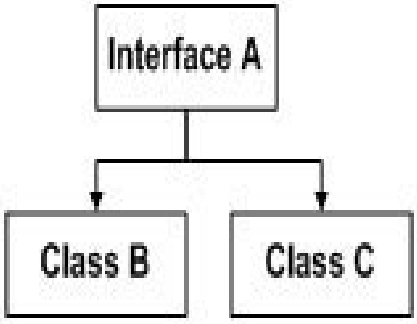
The various forms in which the interfaces can be implemented is described as under:



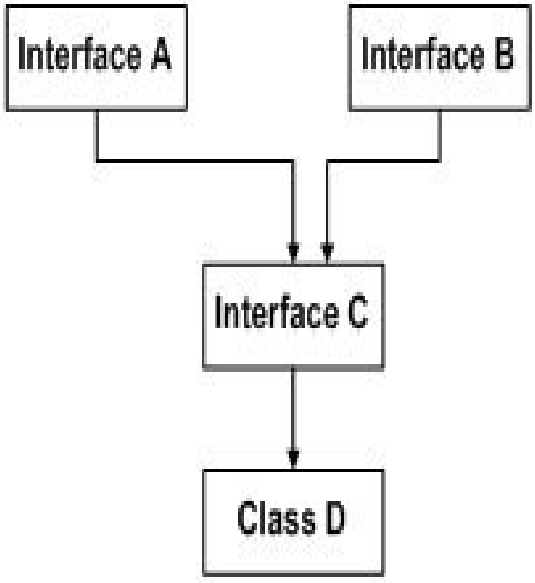
i) An interface A implemented by class B and further extended by class C



ii) An interface E extends interface D and implemented by class B and further extended by class C



iii) An interface A being implemented by class B and class C



i) An interface A an B implemented by interface C and further extended by class D

Figure 4.7: Extending interfaces

So the interface can be implemented by a group of classes and can extend other interface. Now to implement the concept of multiple inheritance, a class cannot extend two classes it has to be a class and implementation of an interface as under:

```
class MyClass extends BaseClass implements Interface1, Interface2…

{

// Body of Myclass with implementation of Interface1 and Interface 2

}
```

It is mandatory for a class to implement all the abstract methods of the interfaces it is implementing. The following examples illustrate the various ways of implementing the interfaces.

This Java program extends one interface from another and the extended interface is implemented by My_Class:

```
package book;

interface IntA

{

void methodIntA();
```

```
int YES =1;
}
interface IntB extends IntA
{
void methodIntB();
int NO=0;
}
class My_Class implements IntB
{
@Override
public void methodIntA() {

System.out.println("You are in method of Interface A & Constant YES =" +YES
);
}

@Override
public void methodIntB() {
System.out.println("You are in method of Interface B & Constant NO =" +NO );
}
}
```

```
public class Interface_Inheritance {
public static void main(String[] args) {
My_Class obj =new My_Class();
obj.methodIntA();
obj.methodIntB();
}
}
```

We will get the following output:

```
You are in method of Interface A & Constant YES =1
You are in method of Interface B & Constant NO =0
```

Understanding the built in interfaces

The Java language comes up with numerous in built interfaces to strengthen its capability to help programmers to write some advanced Java projects, Be it event handling, multithreading and any other functionality of Java, the programmer is provided with interfaces so as it give them more programmatic flexibilities. Some of the inbuilt interfaces with the set of abstract methods are as under:

The Runnable interface is used to implement the Multithreading (To be discussed in upcoming chapter) in Java. The method that needs to be implemented is:

```
public void run();
```

The KeyListener interface is used to handle the keyboard events in Java(To be discussed in Chapter 10: Event Handling in Java). The abstract methods of this interface are:

```
public void keyPressed(KeyEvent arg0)

public void keyReleased(KeyEvent arg0)

public void keyTyped(KeyEvent arg0)
```

The MouseListener interface is used to handle mouse events in Java. The abstract methods are :

public void mouseClicked(MouseEvent arg0)

public void mouseEntered(MouseEvent arg0)

public void mouseExited(MouseEvent arg0)

public void mousePressed(MouseEvent arg0)

public void mouseReleased(MouseEvent arg0)

The below example makes use of the inbuilt interfaces in Java. In the following Java program the MouseListener and KeyListener interfaces are used

```
package book;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;


interface IntA

{

void methodIntA();
```

```
int YES =1;
}
interface IntB extends IntA
{
void methodIntB();
int NO=0;
}
class My_Class implements IntB, KeyListener, MouseListener
{
@Override
public void methodIntA() {

System.out.println("You are in method of Interface A &Constant YES =" +YES
);
}

@Override
public void methodIntB() {
System.out.println("You are in method of Interface B &Constant NO =" +NO );
}
@Override
```

```
public void keyPressed(KeyEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void keyReleased(KeyEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void keyTyped(KeyEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseEntered(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseExited(MouseEvent arg0) {
```

```
// TODO Auto-generated method stub
}
@Override
public void mousePressed(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseReleased(MouseEvent arg0) {
// TODO Auto-generated method stub
}
}
public class Interface_Inheritance {
public static void main(String[] args) {
My_Class obj =new My_Class();
obj.methodIntA();
obj.methodIntB();
}
}
```

We will get the following output:

You are in method of Interface A &Constant YES =1

You are in method of Interface B &Constant NO =0

Assignments

How do we design our own package. How can we make our class available for other programs?

What are the inbuilt packages in Java? Use any class of java.util in your own program?

What are the various access protection available for packages?

What are the various ways of using an interface. How is multiple inheritance implemented in Java? Explain with example?

What are the various ways of importing a package. Why to use a fully qualified name while importing a class?

Create the following packages with different classes mentioned in the table below?

Package Name	Classes
Student	Student BCA IT
Employee	Technical Non-technical

CHAPTER 5

Understanding Arrays, Strings, and Wrapper Classes

Introduction

An array is a collection of similar-typed variables with the intention of referring them by a general name which is common for all the variables. Arrays of every type can be formed and may possibly have one or more dimensions. A precise component in an array is accessed by its index. Arrays put forward a suitable way of grouping associated information. Suppose we are asked to write a program in which we have to create 100 student roll numbers. One option is to create 100 int variables and that is not the recommended approach as it will make a very complicated program. In order to write the concise and efficient programs, the alternative approach is to create an array which is a group of contiguous or related data items that share a common name. For instance we can define an int array roll to represent a set of roll numbers of 100 different students. If we want to access any of the roll number, we have to use the index or subscript in brackets after the array name as:

roll[60]; // represents the roll number of 60th student

With the help of a single looping structure we can access all the elements of an array by changing the index value within the loop. In this chapter we will discuss the one dimensional and two dimensional arrays in detail and we will also focus on strings and vectors in detail.

Structure

Introduction

Implementation of 1-D arrays

Implementation of 2-D arrays

String handling in Java

StringBuffer class

Type wrappers in Java

Assignments

Objective

This chapter explains Arrays in detail. The String handling and Wrapper classes are also explained in this chapter.

Implementation of 1-D Arrays

The one dimensional arrays in Java are declared with one of the following ways:

```
type arrayname[];                          // Declaration
arrayname = new type[size];                // Creation
```

OR

```
type [] arrayname;                         // Declaration
arrayname = new type[size];                // Creation
```

OR

```
type arrayname[] = new type[size];         // Declaration & Creation
```

For example if we want to create an array of roll the following statements are used:

```
int roll[];
roll= new int[5];
```

After creation of an array, we can store the numbers using the statements:

```
roll[0]=50;
roll[1]= 130;
roll[2]= 430;
roll[3]= 300;
roll[4]= 60;
```

Or alternatively, the above assignment can be done by the following statement:

```
int roll[] = {50, 130,430,300,60};
```

This statement will automatically create an array of 5 variables with roll[0] storing the first number that is 50. As the numbers are stored at contiguous memory locations the memory representation is as under:

roll[0]

roll[1]

roll[2]

roll[3]

roll[4]

This Java program sorts a given array of roll numbers stored in an array roll:

```
package book;
public class Array_Sort {
public static void main(String[] args) {
int roll[] = new int[10];
roll[0]=4;
roll[1]=2;
roll[2]=1;
roll[3]=5;
roll[4]=3;
System.out.println("Before Sorting");
for(int i=0;i<5;i++)
System.out.print(roll[i] +" ");
for(int i=0;i<5;i++)
{
for(int j=i+1;j<5;j++)
{
if (roll[i]>roll[j])
{
```

```
int temp=roll[i];
roll[i]=roll[j];
roll[j]=temp;
}
}
}
System.out.println("\n After Sorting");
for(int i=0;i<5;i++)
System.out.print(roll[i] +" ");
}
}
```

We will get the following output:

Before Sorting

4 2 1 5 3

After Sorting

1 2 3 4 5

The following Java program creates an array of N player objects and sorts them based on their score:

```
package book;
class Player
{
int player_id;
String name;
Player()
{}
Player(int a, String s)
{
player_id=a;
name=s;
}
void display()
{
System.out.print(player_id + " " + name);
}
}

class Performance extends Player
{
```

```
int score;
Performance()
{}
Performance(int a, String s, int m)
{
super(a,s);
score=m;

}
void display()
{
super.display();
System.out.println(" " +score );

}
staticvoid sort(Performance a[], int n)
{
Performance temp = new Performance();
for(int i=0;i<n;i++)
{
for(int j=(i+1);j<n;j++ )
```

```
{
if(a[i].score<a[j].score)
{
temp = a[i];
a[i]= a[j];
a[j]=temp;
}
}
}
}
}
public class Player_Sort {

public static void main(String[] args) {
Performance a[] = new Performance[5];

a[0] = new Performance(2,"Saleem", 2000);
a[1] = new Performance(3,"Feroz", 2100);
a[2] = new Performance(4,"Wahid", 1000);
a[3] = new Performance(5,"Nisar", 1500);
a[4] = new Performance(6,"Ishfaq", 1200);
```

```
System.out.println("Before Sorting ");
System.out.print("ID Name Score \n");
for(int i=0;i<5;i++)
a[i].display();
Performance.sort(a, 5);
System.out.println("After Sorting based on Score");
for(int i=0;i<5;i++)
a[i].display();
}
}
```

We will get the following output:

Before Sorting

ID Name Score

2 Saleem 2000

3 Feroz 2100

4 Wahid 1000

5 Nisar 1500

6 Ishfaq 1200

After Sorting based on Score

3 Feroz 2100

2 Saleem 2000

5 Nisar 1500

6 Ishfaq 1200

4 Wahid 1000

In the below Java program, various operations are performed on the array like finding the sum, minimum, maximum and reversing the array. Finally the array is sorted:

```
package book;

public class Array_Operations {


public static void main(String[] args) {

int arr[] = { 20,50,10,60,70};

int sum=0,max,min;

for(int i=0;i<arr.length;i++)

{

sum=sum + arr[i];

}

max=min=arr[0];

System.out.println("The various operations on this array" );
```

```
for(int i=0;i<arr.length;i++)
{
System.out.print(" " + arr[i] );
if (max<arr[i])
max=arr[i];
if (min>arr[i])
min = arr[i];
}
System.out.println("");
System.out.println("The sum of above array =" + sum);
System.out.println("The Max of above array =" + max);
System.out.println("The Min of above array =" + min);
System.out.println("The Reverse of above array =");
for(int i=arr.length-1;i>0;i--)
{
System.out.print(" " + arr[i] );
}
for(int i=0;i<arr.length;i++)
{
for(int j=i+1;j<arr.length;j++)
{
```

```
if (arr[i]>arr[j])
{
int temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
}
}
}
System.out.println("");
System.out.println("After Sorting " );
for(int i=0;i<arr.length;i++)
{
System.out.print(" " + arr[i] );
}
}
}
```

We will get the following output:

The various operations on this array

20 50 10 60 70

The sum of above array =210

The Max of above array =70

The Min of above array =10

The Reverse of above array =

70 60 10 50

After Sorting

10 20 50 60 70

Implementation of 2-D arrays

A two dimensional array is actually the array of array. If suppose, we want to store the information about the air fare from two different cities, we need to use the two dimensional array with two indexes to control the row and column. The following table gives the air fare rates from three different cities to other cities:

	Srinagar	Jammu	Delhi
Mumbai	5000	6000	3000
Bangalore	7000	8000	5000
Ahmadabad	6000	7000	4000

If we want to store this information and process it through our Java program then we need to create a two dimensional array. The syntax for declaration and creation of 2-D array is as under:

```
type arrayname[][];                          // Declaration

arrayname = new type[size][size];            // Creation
```

OR

```
type [][] arrayname;                         // Declaration

arrayname = new type[size][size];            // Creation
```

OR

```
type arrayname[][] = new type[size][size];   // Declaration & Creation
```

For example if we want to create a 2-D array for the above fare rates then the following statements are used:

```
int fare[][];

fare= new int[5][5];
```

After creation of the array we can store the numbers using the statements:

```
fare [0][0]=5000;

fare [0][1]= 6000;

fare [0][2]= 3000;

fare [1][0]= 7000;

fare [1][1]= 8000;

fare [1][2]= 5000;

fare [2][0]= 6000;

fare [2][1]= 7000;

fare [2][2]= 4000;
```

Alternatively, the above statements can be combined into a single statement as under

```
int fare[][] = {

{ 5000,6000,3000},

{ 7000,8000,5000},

{ 6000,7000,4000},

};
```

As the numbers are stored at contiguous memory locations the memory representation is as under. As two indexes are used one controls the row and the other one is used to control the column:

```
fare[row][column ];

[0][0] [0][1] [0][2]

[1][0] [1][1] [1][2]

[2][0] [2][1] [2][2]
```

The below Java programs illustrates the use of two dimensional arrays. The following Java program calculates the addition of two matrices. The two dimensional arrays are used to store the three matrices:

```
package book;

public class Matrix_Addition {


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

```
int a[][]= { {22,55,50,4,51},
{10,2,8,4,9},
{11,2,6,7,2},
{1,20,13,4,15},
};

int b[][]= { {2,5,0,4,1},
{10,12,18,14,19},
{10,7,6,70,20},
{11,10,13,40,10},
};
int c[][]=new int[5][5];
for(int i=0;i<4;i++)
{
for (int j=0;j<5;j++)
{
c[i][j]=a[i][j] + b[i][j];

}
}
System.out.println("The addition of Matrices");
```

```
for(int i=0;i<4;i++)
{
for (int j=0;j<5;j++)
{
System.out.print(c[i][j] + " ");
}
System.out.println();
}
}
}
```

We will get the following output:

The addition of Matrices

24 60 50 8 52

20 14 26 18 28

21 9 12 77 22

12 30 26 44 25

The following program helps you to create a multiplication table with the help of 2-D arrays:

```
package book;
public class MultiTable {
public static void main(String[] args) {
int m[][] = new int[11][11];
int i,j;
System.out.println("Multiplication Table");
System.out.print(" " );
for(i=1;i<=10;i++)
System.out.print(" " + i);
System.out.println();
for(i=1;i<=10;i++)
{
System.out.print(i + " ");
for(j=1;j<=10;j++)
{
m[i][j]= i*j;
System.out.print(" " + m[i][j]);
}
System.out.println();
}
```

```
}
}
```

We will get the following output:

Multiplication Table

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

In the belowJava program, the multiplication of two matrices is performed:

```
package book;
public class Matrix_Multiplication {
public static void main(String[] args) {
int a[][]= { {1,2},
{2,2},
};
int b[][]= { {2,5},
{1,2},
};
int c[][] = new int[5][5];

for(int i=0;i<2;i++)
{
for (int j=0;j<2;j++)
{
for (int k=0;k<2;k++)
{
c[i][j]=c[i][j] + a[i][k]*b[k][j];
```

```
}
}
}
System.out.println("The 1st Matrix =");
for(int i=0;i<2;i++)
{
for (int j=0;j<2;j++)
{
System.out.print(a[i][j] + " ");
}
System.out.println("");
}

System.out.println("The 2nd Matrix =");
for(int i=0;i<2;i++)
{
for (int j=0;j<2;j++)
{
System.out.print(b[i][j] + " ");
}
System.out.println("");
```

```
}
System.out.println("The Multiplication of Matrices");
for(int i=0;i<2;i++)
{
for (int j=0;j<2;j++)
{
System.out.print(c[i][j] + " ");
}
System.out.println();
}
}
}
```

We will get the following output:

```
The 1st Matrix =

1   2

2   2

The 2nd Matrix =

2   5

1   2

The Multiplication of Matrices

4   9

6   14
```

String handling in Java

String handling is the major task in any of the project development. Strings are the group of characters which can also be handled by creating a character array as is done in many of the programming languages but Java gives special importance to string handling and implements it in the form of a class called as the String class. So a string in Java is not a primitive data type but a complete class. So if we talk about a class it has a complete set of constructors and methods to implement different set of functionalities. The String class has a set of methods to compare two strings, search for a character within a string, find a substring within a string, and change the case of letters to upper and lower cases within a string. Also, String objects can be constructed in a number of ways, making it straightforward to get a string when desired.

The String, StringBuffer, and StringBuilder classes are defined in java.lang package. Thus, being a part of the default package, they are available to all programs without import statement. All are declared final, which means that no one of these classes may be inherited.

To create a String object, one of the following constructors can be used:

String();

String(char ch[]);

String(char ch, int startindex, int numchars);

String(String strobj);

String(byte asciichars[]);

String(byte asciichars[],int startindex, int numchars);

Now the following example illustrates the creation of String object using the constructors. In the below Java program, String objects are created with different constructors of string class:

```
package book;

public class String_Demo {

public static void main(String[] args) {

System.out.println("Creation of String object using String Constructors");

String str1 = new String();

char c[]= { 'N','I','E','L','I','T'};

String str2 = new String(c);

String str3 = new String(c, 4,2);

String str4 = new String(str2);

byte b[]={65,66,67,68};

String str5 = new String(b);

String str6 = new String(b,0,2);

System.out.println("Using Default Constructor str1 holds nothing " + str1);

System.out.println("str2 = " +str2);

System.out.println("str3 = " +str3);

System.out.println("str4 = " +str4);
```

```
System.out.println("str5 = " +str5);

System.out.println("str6 = " +str6);

}

}
```

We will get the following output:

Creation of String object using String Constructors

Using Default Constructor str1 holds nothing

str2 = NIELIT

str3 = IT

str4 = NIELIT

str5 = ABCD

str6 = AB

The various string methods which can be easily accessed by the intelligence of Eclipse editor by pressing the object name and dot operator or Ctrl+ Spacebar. The following table give us the details of various methods available in String class:

Method
int length( )

char charAt(int where)
void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)
byte[ ] getBytes( )
char[ ] toCharArray( )
boolean equals(Object str)
boolean equalsIgnoreCase(String str)
boolean regionMatches(int startIndex, String str2,int str2StartIndex, int numCha
boolean regionMatches(boolean ignoreCase,int startIndex, String str2,int str2Sta
boolean startsWith(String str)
boolean endsWith(String str)
boolean startsWith(String str, int startIndex)
int compareTo(String str)
int compareToIgnoreCase(String str)
int indexOf(int ch)
int lastIndexOf(int ch)
int indexOf(String str)
int lastIndexOf(String str)
int indexOf(int ch, int startIndex)
int lastIndexOf(int ch, int startIndex)
int indexOf(String str, int startIndex)
int lastIndexOf(String str, int startIndex)
String substring(int startIndex)
String substring(int startIndex, int endIndex)
String concat(String str)
String replace(char original, char replacement)
String replace(CharSequence original, CharSequence replacement)

String trim( )
static String valueOf(double num)
static String valueOf(long num)
static String valueOf(Object ob)
static String valueOf(char chars[ ])
static String valueOf(char chars[ ], int startIndex, int numChars)
String toLowerCase( )
String toUpperCase( )
int codePointAt(int i)
int codePointBefore(int i)
int codePointCount(int start, int end)
boolean contains(CharSequence str)
boolean contentEquals(CharSequence str)
boolean contentEquals(StringBuffer str)
static String format(String fmtstr, Object … args)
static String format(Locale loc, String fmtstr,Object … args)
boolean matches(string regExp)
int offsetByCodePoints(int start, int num)
String replaceFirst(String regExp, String newStr)
String replaceAll(String regExp, String newStr)

Table 5.1: String methods

Let us understand some of the important methods available in String class.

int compareTo(String str2)

This method compares the invoking string with the str2 object of String class and returns an int value which can have zero, negative or positive values. if zero then two strings are equal, if less than zero then invoking string is less than str2 and if greater than zero then invoking string is greater than str2. If for instance the following segment of program, we can use this method to compare two String objects:

```
String str1= new String("Java");
String str2= new String("Hello");
if (str1.compareTo(str2)>0)
System.out.println(" Str1 is Greater");
else
System.out.println(" Str2 is Greater");
```

The following example program makes use of the methods available in string class. The various methods of String class are used in this program to explain the implementation of string handling in Java:

```
package book;
public class String_Handling {
public static void main(String[] args) {
String str = new String("I Love Java");
System.out.println("The Input String is----> " + str);
System.out.println("The Length of above String = " + str.length());
System.out.println("The Character at 0th Position = " + str.charAt(0));
System.out.println("The SubString = " + str.substring(2, 6));
System.out.println("Coverted to UpperCase " + str.toUpperCase());
System.out.println("After Concat----> " + str.concat(" and Android"));
System.out.println("After getBytes " + str.getBytes().toString());
}
}
```

We will get the following output:

The Input String is----> I Love Java

The Length of above String = 11

The Character at 0th Position = I

The SubString = Love

Coverted to UpperCase I LOVE JAVA

After Concat----> I Love Java and Android

After getBytes [B@7852e922

In the belowJava program, N strings are given as command line arguments. The strings are sorted using the string methods.

```
package book;

public class String_Sort {

public static void main(String[] args) {

System.out.println("Before Sorting");

for(int i=0;i<args.length;i++)

{

System.out.print(args[i] + " ");

}

for(int i=0;i<args.length;i++)

{

for(int j=i+1;j<args.length;j++)

{

if (args[i].compareTo(args[j])>0)

{
```

```
String temp= args[i];
args[i]= args[j];
args[j]=temp;
}
}
}
System.out.println("");
System.out.println("After Sorting");
for(int i=0;i<args.length;i++)
{
System.out.print(args[i] + " ");
}
}
}
```

We will get the following output:

Before Sorting

Riyan Maheen Barka Aysha Mayra

After Sorting

Aysha Barka Maheen Mayra Riyan

StringBuffer class

StringBuffer is a class with some additional capabilities as compared to a String class. String class characterize a fixed-length, absolute character sequences. In distinction, StringBuffer correspond to extendable and writeable character sequences. StringBuffer may have characters and substrings added in the center or attached to the end. StringBuffer will by design grow to construct space for such add-ons and frequently has additional characters pre-allocated than are truly desired, to permit space for enlargement.

The constructors used for creation of StringBuffer object are as under:

Constructor	Description
StringBuffer( )	The default constructor reserves room for 16
StringBuffer(int size)	Explicitly sets the size of the buffer
StringBuffer(String str)	Accepts a String argument that sets the initia
StringBuffer(CharSequence chars)	creates an object that contains the character s

Table 5.2: Constructor methods of StringBuffer

The following methods are available in the StringBuffer class:

Method	Desc
String substring(int startIndex)	Retu
String substring(int startIndex, int endIndex)	Retu
StringBuffer replace(int startIndex, int endIndex, String str)	Repl
StringBuffer delete(int startIndex, int endIndex)	Delet
StringBuffer deleteCharAt(int loc)	Delet
StringBuffer reverse( )	Reve
StringBuffer insert(int index, String str)	Inser
StringBuffer insert(int index, char ch)	Inser
StringBuffer insert(int index, Object obj)	Inser
StringBuffer append(String str)	Conc
StringBuffer append(int num)	Conc
StringBuffer append(Object obj)	Conc
void getChars(int sourceStart, int sourceEnd, char target[ ], int targetstart)	Copi
char charAt(int where)	Gets
void setCharAt(int where, char ch)	Sets t
void setLength(int len)	Sets t
void ensureCapacity(int capacity)	Sets t
int length( )	Gets

int capacity( )	The t
StringBuffer appendCodePoint(int ch)	Appe
int codePointAt(int i)	Retur
int codePointBefore(int i)	Retur
int codePointCount(int start, int end)	Retur
int indexOf(String str)	Searc
int indexOf(String str, int startIndex)	Searc
int lastIndexOf(String str)	Searc
int lastIndexOf(String str, int startIndex)	Searc

Table 5.3: Methods of StringBuffer class

The following programs illustrates the usage of methods in the StringBuffer class:

```
package book;

class StringBufferMethods {

public static void main(String args[])

{

StringBuffer sb_obj = new StringBuffer("NIELIT");

System.out.println("Buffer = " + sb_obj);

System.out.println("Length of Object= " + sb_obj.length());

System.out.println("Capacity of Object " + sb_obj.capacity());

sb_obj.ensureCapacity(100);

System.out.println("Now Capacity = " + sb_obj.capacity());

System.out.println("Character At 0th Location = " + sb_obj.charAt(0));

sb_obj.setCharAt(0, 'R');

System.out.println("Buffer = " + sb_obj);

char ch[] = new char[10];

sb_obj.getChars(0, 3, ch, 0);
```

```
System.out.println("ch = " + ch[0] + ch[1] + ch[2]);

System.out.println("Append = " + sb_obj.append("J&K"));

System.out.println("Insert = " + sb_obj.insert(0, ch));

System.out.println("Reverse = " + sb_obj.reverse());

System.out.println("Replace = " + sb_obj.replace(0, 2, "OK"));

System.out.println("Sub String = " + sb_obj.substring(0, 2));

}

}
```

We will get the following output:

Buffer = NIELIT

Length of Object= 6

Capacity of Object 22

Now Capacity = 100

Character At 0th Location = N

Buffer = RIELIT

Append = RIELITJ&K

Insert = RIE RIELITJ&K

Reverse = K&JTILEIR EIR

Replace = OKJTILEIR EIR

Sub String = OK

Wrapper classes

In addition to the primitive data types like int, float, char, and more Java uses built in classes called the type wrappers which correspond to the basic data types to encapsulate the functionalities of these basic data types. The classes like Integer, Float, Character, and many more are fully equipped with methods to deal with the basic functionalities related to data. All these wrapper classes are defined in java.lang package and are imported automatically to all the Java programs.

The abstract class Number defines a superclass that is implemented by the classes that wrap the numeric types byte, short, int, long, float, and double.

The various methods available in Number class are:

Method	Description
byte byteValue( )	Returns the value as byte
int intValue( )	Returns the value as int
double doubleValue( )	Returns the value as double
long longValue( )	Returns the value as long
float floatValue( )	Returns the value as float
short shortValue( )	Returns the value as short

Table 5.4: Methods in Number class

Double and Float

Double and float are wrappers for floating-point values of type double and float, respectively.

The constructors for Float are shown here:

Float(double num)

Float(float num)

Float(String str) throws NumberFormatException

Double(double num)

Double(String str) throws NumberFormatException

The methods defined by float are shown below:

Method	Description
byte byteValue( )	Returns the v
static int compare(float num1, float num2)	Compares th
int compareTo(Float f)	Compares th
double doubleValue( )	Returns the v

boolean equals(Object FloatObj)	Returns true
static int floatToIntBits(float num)	Returns the I
static int floatToRawIntBits(float num)	Returns the I
float floatValue( )	Returns the v
int hashCode( )	Returns the h
static float intBitsToFloat(int num)	Returns float
int intValue( )	Returns the v
boolean isInfinite( )	Returns true
static boolean isInfinite(float num)	Returns true
boolean isNaN( )	Returns true
static boolean isNaN(float num)	Returns true
long longValue( )	Returns the v
static float parseFloat(String str) throws NumberFormatException	Returns the f
short shortValue( )	Returns the v
static String toHexString(float num)	Returns a stri
String toString( )	Returns the s
static String toString(float num)	Returns the s
static Float valueOf(float num)	Returns a Flo
static Float valueOf(String str) throws NumberFormatException	Returns the F

Table 5.5: The methods defined by Float

The methods for Double are shown here:

Method defined by Double	Descript
byte byteValue( )	Returns
static int compare(double num1, double num2)	Compare
int compareTo(Double d)	Compare
double doubleValue( )	Returns
boolean equals(Object DoubleObj)	Returns
float floatValue( )	Returns
int hashcode( )	Returns
int intValue( )	Returns
boolean isInfinite( )	Returns
static boolean isInfinite(double num)	Returns
boolean isNaN( )	Returns
static boolean isNaN(double num)	Returns
static double longBitsToDouble(long num)	Returns
long longValue( )	Returns
static double parseDouble(String str) throws NumberFormatException	Returns
short shortValue( )	Returns
static String toHexString(double num)	Returns
String toString( )	Returns

static String toString(double num)	Returns
static Double valueOf(double num)	Returns
static Double valueOf(String str) throws NumberFormatException	Returns

Table 5.6: Methods defined by Double

Byte, Short, Integer, and Long

The Byte, Short, Integer, and Long classes are wrappers for byte, short, int, and long integer types, respectively. Their constructors are shown here:

Byte(byte num)

Byte(String str) throws NumberFormatException

Short(short num)

Short(String str) throws NumberFormatException

Integer(int num)

Integer(String str) throws NumberFormatException

Long(long num)

Long(String str) throws NumberFormatException

The various methods available in Byteclass are as under:

Method	Des
byte byteValue( )	Retu
int compareTo(Byte b)	Com
static Byte decode(String str) throws NumberFormatException	Retu
double doubleValue( )	Retu

boolean equals(Object ByteObj)	Retu
float floatValue( )	Retu
int hashCode( )	Retu
int intValue( )	Retu
long longValue( )	Retu
static byte parseByte(String str) throws NumberFormatException	Retu
static byte parseByte(String str, int radix) throws NumberFormatException	Retu
short shortValue( )	Retu
String toString( )	Retu
static String toString(byte num)	Retu
static Byte valueOf(byte num)	Retu
static Byte valueOf(String str) throws NumberFormatException	Retu
static Byte valueOf(String str, int radix) throws NumberFormatException	Retu

Table 5.7: The methods defined by Byte

The various methods available inShort class are as under:

Method	De
byte byteValue( )	Re
int compareTo(Short s)	Co
static Short decode(String str) throws NumberFormatException	Re
double doubleValue( )	Re
boolean equals(Object ShortObj)	Re
float floatValue( )	Re
int hashCode( )	Re
int intValue( )	Re
long longValue( )	Re
static short parseShort(String str) throws NumberFormatException	Re
static short parseShort(String str, int radix) throws NumberFormatException	Re
static short reverseBytes(short num)	Ex
short shortValue( )	Re
String toString( )	Re
static String toString(short num)	Re
static Short valueOf(short num)	Re
static Short valueOf(String str) throws NumberFormatException	Re
static Short valueOf(String str, int radix) throws NumberFormatException	Re

Table 5.8: The methods defined by Short

The various methods available in Integer class are as under:

Method	Description
static int bitCount(int num)	Returns the nu
byte byteValue( )	Returns the va
int compareTo(Integer i)	Compares the
static Integer decode(String str) throws NumberFormatException	Returns an Int
double doubleValue( )	Returns the va
boolean equals(Object IntegerObj)	Returns true i
float floatValue( )	Returns the va
static Integer getInteger(String propertyName)	Returns the va
static Integer getInteger(String propertyName, int default)	Returns the va
static Integer getInteger(String propertyName, Integer default)	Returns the va
int hashCode( )	Returns the ha
static int highestOneBit(int num)	Determines th
int intValue( )	Returns the va
long longValue( )	Returns the va
static int lowestOneBit(int num)	Determines th
static int numberOfLeadingZeros(int num)	Returns the nu

Table 5.9: The methods defined by Integer

The various methods available in Integer class are as under:

Method	Des
static int numberOfTrailingZeros(int num)	Ret
static int parseInt(String str) throws NumberFormatException	Ret
static int parseInt(String str, int radix) throws NumberFormatException	Ret
static int reverse(int num)	Rev
static int reverseBytes(int num)	Rev
static int rotateLeft(int num, int n)	Ret
static int rotate Right(int num, int n)	Ret
static int signum(int num)	Ret
short shortValue( )	Ret
static String toBinaryString(int num)	Ret
static String toHexString(int num)	Ret
static String toOctalString(int num)	Ret
String toString( )	Ret
static String toString(int num)	Ret
static String toString(int num, int radix)	Ret
static Integer valueOf(int num)	Ret
static Integer valueOf(String str) throws NumberFormatException	Ret
static Integer valueOf(String str, int radix) throws NumberFormatException	Ret

Table 5.10: The methods defined by Integer

The various methods available in Long class are as under:

Method	Des
static int bitCount(long num)	Ret
byte byteValue( )	Ret
int compareTo(Long l)	Cor
static Long decode(String str) throws NumberFormatException	Ret
double doubleValue( )	Ret
boolean equals(Object LongObj)	Ret
float floatValue( )	Ret
static Long getLong(String propertyName)	Ret
int hashCode( )	Ret
static long highestOneBit(long num)	Det
int intValue( )	Ret
long longValue( )	Ret
static long lowestOneBit(long num)	Det
static long parseLong(String str) throws NumberFormatException	Ret
static long parseLong(String str, int radix) throws NumberFormatException	Ret
static long reverse(long num)	Rev
static long reverseBytes(long num)	Rev
static long rotateLeft(long num, int n)	Ret

static long rotateRight(long num, int n)	Ret
static int signum(long num)	Ret
short shortValue( )	Ret
static String toBinaryString(long num)	Ret
static String toHexString(long num)	Ret
static String toOctalString(long num)	Ret
String toString( )	Ret
static String toString(long num)	Ret
static String toString(long num, int radix)	Ret
static Long valueOf(long num)	Ret
static Long valueOf(String str) throws NumberFormatException	Ret
static Long valueOf(String str, int radix) throws NumberFormatException	Ret

Table 5.11: Methods defined by Long

Character

Character is a straightforward wrapper around a char. The constructor for Character is as follows:

Character(char ch)

The various methods available in Character class are as under:

Method	Description
static boolean isDefined(char ch)	Returns true if ch is defined by
static boolean isDigit(char ch)	Returns true if ch is a digit. Oth
static boolean isIdentifierIgnorable(char ch)	Returns true if ch should be ign
static boolean isISOControl(char ch)	Returns true if ch is an ISO cor
static boolean isJavaIdentifierPart(char ch)	Returns true if ch is allowed as
static boolean isJavaIdentifierStart(char ch)	Returns true if ch is allowed as
static boolean isLetter(char ch)	Returns true if ch is a letter. Ot
static boolean isLetterOrDigit(char ch)	Returns true if ch is a letter or a
static boolean isLowerCase(char ch)	Returns true if ch is a lowercas
static boolean isMirrored(char ch)	Returns true if ch is a mirrored
static boolean isSpaceChar(char ch)	Returns true if ch is a Unicode
static boolean isTitleCase(char ch)	Returns true if ch is a Unicode
static Boolean isUnicodeIdentifierPart(char ch)	Returns true if ch is allowed as
static Boolean isUnicodeIdentifierStart(char ch)	Returns true if ch is allowed as

static boolean isUpperCase(char ch)	Returns true if ch is an upperca
static boolean isWhitespace(char ch)	Returns true if ch is whitespace
static char toLowerCase(char ch)	Returns lowercase equivalent o
static char toTitleCase(char ch)	Returns titlecase equivalent of
static char toUpperCase(char ch)	Returns uppercase equivalent o

Table 5.12: Various Character methods

Boolean

Boolean is a very thin wrapper around boolean values, which is helpful generally when we want to pass a boolean variable by reference. It contains the constants TRUE and FALSE, which define true and false Boolean objects. Boolean also defines the TYPE field, which is the Class object for Boolean. Boolean defines these constructors:

Boolean(boolean boolValue)

Boolean(String boolString)

The various methods available in Boolean class are as under:

Method	Description
boolean booleanValue( )	Returns boolean equivalent.
int compareTo(Boolean b)	Returns zero if the invoking ol
boolean equals(Object boolObj)	Returns true if the invoking ob
static boolean getBoolean(String propertyName)	Returns true if the system prop
int hashCode( )	Returns the hash code for the i
static boolean parseBoolean(String str)	Returns true if str contains the
String toString( )	Returns the string equivalent c
static String toString(boolean boolVal)	Returns the string equivalent c
static Boolean valueOf(boolean boolVal)	Returns the Boolean equivaler

static Boolean valueOf(String boolString)	Returns true if boolString cont

Table 5.13: Methods defined by Boolean

The below program illustrates the usage of Type Wrappers. In this Java program, one of the important methods of Integer is demonstrated. The command line arguments are supplied as: 1 Shahid 400 2 Rahid 600 3 Majid 500. As all the arguments are in String format, the Integer.parseInt method is used to covert the string integers to int data type:

```
package abc;

class Candidate

{

int roll;

String name;

Candidate()

{}

Candidate(int a, String s)

{

roll=a;

name=s;

}

void display()

{
```

```
System.out.print(roll + " " + name);
}
}
interface SportsMarks
{
static final int sp_marks=50;
void show_sports();
}
class Topper extends Candidate implements SportsMarks
{
int marks;
Topper()
{}
Topper(int a, String s, int m)
{
super(a,s);
marks=m + sp_marks;
}
void display()
{
super.display();
```

```
System.out.println(" " +marks );
}
staticvoid sort(Topper a[], int n)
{
Topper temp = new Topper();
for(int i=0;i<n;i++)
{
for(int j=(i+1);j<n;j++ )
{
if(a[i].marks<a[j].marks)
{
temp = a[i];
a[i]= a[j];
a[j]=temp;
}
}
}
}
@Override
public void show_sports() {
System.out.println("Sports Marks = " + sp_marks);
```

```
}
}
public class WrapperDemo {
public static void main(String[] args) {
Topper a[] = new Topper[5];
int i=0,j=1,k=2,l;
for (l=0;l<3;l++)
{
a[l]=new Topper(Integer.parseInt(args[i]),args[j], Integer.parseInt(args[k]));
i=i+3;
j=j+3;
k=k+3;
}

System.out.println("Before Sorting ");
for(l=0;l<3;l++)
a[l].display();
Topper.sort(a, 3);
System.out.println("After Sorting based on Marks and Sports marks");
for(l=0;l<3;l++)
a[l].display();
```

```
}
}
```

We will get the followingoutput:

Before Sorting

1 Shahid 450

2 Rahid 650

3 Majid 550

After Sorting based on Marks and Sports marks

2 Rahid 650

3 Majid 550

1 Shahid 450

In the following Java program, the Character class is explored:

```
package abc;
public class CharWrapperDemo {
public static void main(String args[])
{
char data[] = {'N', 'i', 'E', 'L', 'I', 'T', ' ', '7'};
```

```
for(int i=0; i<data.length; i++) {
if(Character.isDigit(data[i]))
System.out.println(data[i] + " is data digit.");
if(Character.isLetter(data[i]))
System.out.println(data[i] + " is data letter.");
if(Character.isWhitespace(data[i]))
System.out.println(data[i] + " is whitespace.");
if(Character.isUpperCase(data[i]))
System.out.println(data[i] + " is uppercase.");
if(Character.isLowerCase(data[i]))
System.out.println(data[i] + " is lowercase.");
}
}
}
```

We will get the following output:

N is data letter.

N is uppercase.

i is data letter.

i is lowercase.

E is data letter.

E is uppercase.

L is data letter.

L is uppercase.

I is data letter.

I is uppercase.

T is data letter.

T is uppercase.

is whitespace.

7 is data digit.

Assignments

Write a Java program that inserts, updates and deletes a particular number from a one dimensional array based on the choice given by the user?

How does String class differ from StringBuffer class?

Write a Java program to merge two sorted arrays arr1 and arr2 into a third array arr3. Display the third array in the ascending order?

What are the various methods available in String class?

Write a program in Java, which will read a text and count all occurrences of a particular word?

CHAPTER 6

Exception Handling in Java

Introduction

One of the challenges for a programmer is to deal with the errors, particularly the run time errors which are raised at the time of execution of the Java program. The reason for the run time errors is because of a miscalculation by the programmer. For example if we have to execute a following line of code after getting the inputs from the user:

Z= ((a*b)-c)/ (b-c);

The programmer may not be sure about the values user is giving as input and if somehow the value of (b-c) comes equal to zero, then our programme will abnormally terminate because we are trying to divide a number by zero which will lead to a divide by zero exception and our program will terminate. One more example with the help of which we can understand the concept of Exception is if we are getting the values from command line and trying to find out whether the number is prime or not. In dealing with such type of program, our program expects a number as command line argument while executing the program. If the number is not provided then it will lead to NumberFormatException.

So a Java exception is an object that portrays an extraordinary situation that has occurred in a portion of code. When an exceptional situation arises, anobject signifying that exception is produced and thrown in the method that is a reason for that error. That method could decide to knob the exception itself, or exceed it on. Whichever way, at some point, the exception is trapped and processed. The Exception is a run time error or an abnormal line of code that arises while executing the Java program. Dealing with those run time errors or exception through Java's exception handling mechanisms is called as exception handling. In this chapter we will use most of the practical examples to understand the

concept of exception handling so as to handle the exceptions properly in our Java projects.

Structure

Introduction

Understanding exception handling

Using try and catch

Nested try…catch statements

Understanding the throw and throws.

Understanding built-in exception classes

Creating user-defined exception

Assignments

Objective

This chapter explains the various types of errors with the focus on Exception handling. The reader of this chapter will be able to handle exceptions in any of the Java program.

Understanding exception handling

In order to understand the concept of exception handling let us execute the below program to find out how the Java's run time environment will react to it. What we are expecting from this program is to output two lines which are Before Exception and After Exception and one basic calculation statement is written in between these two lines:

```
public class Exception_Demo {

public static void main(String[] args) {

int a,b,c,z;

a=10;

b=10;

c=10;

System.out.println("Before Exception");

z=(a+b)/(b-c);

System.out.println("After Exception");

}

}
```

Now we will run this program and the following output is generated:

Before Exception

Exception in thread "main" java.lang.ArithmeticException: / by zero at book. Exception_Demo.main(Exception_Demo.java:10)

If we carefully look at the output generated by the above program the first line was generated that is "Before Exception " but something has went wrong in the following line:

z=(a+b)/(b-c);

In this mathematical calculation we are trying to divide a number by zero and that has led to the abnormal termination of our program. That means as a programmer we must understand were the exception can get generated and we must handle that within the program. Now the question is how to deal or handle such errors? For doing that we must understand the output generated by the above program. It generates an object of java.lang.ArithematicException class and that object we must handle within the programme using try, catch, and finally blocks to be discussed in the next section.

We will take one more example to understand the concept of exception handling:

```
package book;

public class Exception_Demo {

public static void main(String[] args) {

int n[]= new int[10];

int sum=0;
```

```
for (int i=0;i<3;i++)

{

n[i]=Integer.parseInt(args[i]);

sum=sum+n[i];

}

System.out.println("The sum =" +sum);

}

}
```

After executing the above program without any command line arguments the following error or exception is generated:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0at book.Exception_Demo.main(Exception_Demo.java:8)

The exception that we have not handled is the ArrayIndexOutOfBoundsException. As a programmer we expected that the user will give input at run time and program will generate the output accordingly as the arguments are not properly given and String args[] array is not initialized.

Using try,catch, and finally

The Java provides an inbuilt mechanism to deal with the errors popping up at run time and abnormally terminating the program. As understood from the previous section, whenever an exception occurs it generates an Exception object with all the details about the error. If we want to catch that object we have to use try, catch, and finally blocks. The syntax is as under:

```
try

{

// Block of code where Exception can Occur

}

catch (ExceptionObject obj)

{

// Error Messages for the User and Corrective Actions

}

finally

{

// This Block is optional and will always get Executed

}
```

In order to use the exception handling the programmer must follow the following

steps:

The programmer must understand the code properly and find out where the exception can occur. He should not use the try and catch block unnecessarily.

The programmer should enclose that part of a code within try block where the exception can generate. If the line or lines of code is properly enclosed in try block it will throw the object to the catch block.

After enclosing the block of code in try block, the programmer has to catch those objects in the catch block.

If a programmer wants to take some corrective steps as he may not be sure whether the exception will occur or not that is whether try block will be completely executed or catch block, he may use finally block which will always get executed.

Exception types

Each and every one exception types are subclasses of the built-in class Throwable. Therefore, Throwable is at the peak of the exception class ladder. Right away beneath Throwable are two subclasses that separate exceptions into two different branches. One limb is headed by Exception. This class is used for exceptional circumstances that user programs must catch. This is furthermore the class that we will subclass to create our own user defined exception types. There is an important subclass of Exception, called RuntimeException. Exceptions of this type are automatically defined for the programs that you write and include things for instance division by zero and invalid array indexing.

In the following Java program, the line in which the exception is expected is enclosed in the try block and as it tries to divide by zero, it generates an object of ArithematicException type and that object is caught in catch block:

```
public class Exception_Demo {

public static void main(String[] args) {

int a,b,c,z;

a=10;

b=10;

c=10;

System.out.println("Before Exception");

try{

z=(a+b)/(b-c);
```

```
}
catch(ArithematicException obj)
{
System.out.println("Divide by Zero Exception");
}
System.out.println("After Exception");
}
}
```

We will get the following output:

Before Exception

Divide by Zero Exception

After Exception

In the following Java program, the statement which can generate the exception at run time is enclosed in try block and as the program depends on the command line arguments to be given by the user. The user may or may not give the exact number of inputs, as such we must use exception handling to deal with such scenarios:

```
package book;
public class Exception_Demo {
```

```
public static void main(String[] args) {
int n[]= new int[10];
int sum=0;
try
{
for (int i=0;i<3;i++)
{
n[i]=Integer.parseInt(args[i]);
sum=sum+n[i];
}
}
catch(ArrayIndexOutOfBoundsException obj)
{
System.out.println("Error in arguments");
}
System.out.println("The sum =" +sum);
}
}
```

We will get the following output:

Error in arguments

The sum =0

In the following program we have used all the three blocks that is the try, catch and finally block. As we are not sure whether try or catch block will be completely executed and if we have to execute some statements in any case then we must use the finally block as it always gets executed. In this Java program we tried to open a file. The file may or may not exist and can generated exception at runtime, so we used the try block while opening the file and used finally block to close the file:

```
package book;

import java.io.*;

public class Finally_Demo {


public static void main(String[] args) {

File f1 = new File("in.dat");

FileInputStream fin=null;

try

{

fin = new FileInputStream(f1);

System.out.println("File OK");

}

catch(IOException e)
```

```
{
System.out.println("Error in Opening a File");
}
finally
{
try{
fin.close();
}
catch(IOException e)
{
System.out.println(e);
}
}
}
}
```

We will get the following output:

Error in Opening a File

Exception in thread "main" java.lang.NullPointerException

at book.Finally_Demo.main(Finally_Demo.java:20)

Nested try…catch statements

In our Java program, we may have a single statement but that may generate multiple exceptions. For example the below statement in Java can be evaluated:

```
sum = a[i]/ b[i+j];
```

In this statement we are dividing a number by some other number and that other number can be zero so one of the exception that may generate is ArithematicException. Now the second exception that may generate is ArrayIndexOutOfBoundsException as we are not sure what the value of (i+j) will be. so for a single statement we may get two exceptions. So we can enclose this statement in tryblock followed by two catch statements:

```
try

{

Statement ;

}

catch(ExceptionType1 e)

{

}

catch(ExceptionType2 e)

{
```

```
}

catch(ExceptionTypeN e)

{

}
```

In the following Java program, we supplied the arguments but the arguments are not in the right format so the exception generated by this program is the NumberFormatException. In this program only one try statement is executed and threecatch blocks are there to catch three different types of exceptions:

```
package book;

public class Exception_Demo {

public static void main(String[] args) {

int n[]= new int[10];

int sum=0;

try

{

for (int i=0;i<3;i++)

{

n[i]=Integer.parseInt(args[i]);

sum=sum+n[i];

}
```

```
}
catch(ArrayIndexOutOfBoundsException obj)
{
System.out.println("Error in arguments");
}
catch(NumberFormatException obj)
{
System.out.println("Arguments are not in right Format");
}
catch(ArithmeticException obj)
{
System.out.println("Divide by Zero Exception");
}
System.out.println("The sum =" +sum);
}
}
```

We will get the following output:

Arguments are not in right Format

The sum =0

In the below program we have used the nested try…catch statements. As two try blocks along with two catch statements are used. This programme executes two catch blocks as two exceptions are thrown:

```
package book;

public class Multiple_Catch {

public static void main(String[] args) {


try

{

int a[]={10};

int b,c,z;

b=10;

c=10;

try {

z= a[0]/(b-c);

}

catch(ArithmeticException e)

{

System.out.println("Division By Zero in Arithmetic Exception");

}
```

```
a[2]=70;
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Array Index out of Bounds Exception");
}
}
}
```

We will get the following output:

Division By Zero in Arithmetic Exception

Array Index out of Bounds Exception

The following program explains the use of nested try blocks:

```
package book;
public class NestedTryExample {
public static void main(String args[]) {
try {
int a = args.length;
```

```
/* If no command-line args are present,
the following statement will generate
a divide-by-zero exception. */
int b = 142 / a;
System.out.println("a = " + a);

try { // nested try block
/* If one command-line arg is used,
then a divide-by-zero exception
will be generated by the following code. */ if(a==1) a = a/(a-a); // division by zero
/* If two command-line args are used,
then generate an out-of-bounds exception. */
if(a==2) {
int c[] = { 1 };
c[35] = 60; // generate an out-of-bounds exception
}
} catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array index out-of-bounds: " + e);
}
}
```

```
catch(ArithmeticException e) {

System.out.println("Divide by Zero: " + e);

}

}

}
```

We will get the following output:

a = 2

Array index out-of-bounds: java.lang.ArrayIndexOutOfBoundsException: 35

Understanding throw and throws

These two keywords in Java have an utmost importance while dealing with exception handling in Java. Sometimes we are in need of explicitly throwing the exception and we can do that by using the throw keyword along with the Exception object that we want to throw. On the other hand the throws is used with the method names where they offload themselves by giving the responsibility to the programmer to handle that exception. These two keywords are explained in detail as under.

throw keyword

As discussed in the previous sections, we understood that it's the exception object that goes to the catch block if any exception occurs. So it all depends on the runtime error, if it occurs then only the object with all the details will be sent to the catch block. What if we want to forcibly take the control to the catch block, at that time we can make use of throw for example:

```
throw new IOException();

throw new NumberFormatException();

throw new ArithmeticException();
```

All the above three statements generate three objects of their respective class and are sent to the catch block. The following examples illustrate throw statement is used to generate the objects and throw them to the catch blocks forcibly. If we go through the program we find that there are no exception as the division operation is correct and so is the usage of array but still we managed to execute the catch block because of the throw statement:

```
package book;

public class Multiple_Catch {

public static void main(String[] args) {

try
```

```
{
int a[]={10};
int b,c,z;
b=100;
c=10;
try {
z= a[0]/(b-c);
System.out.println("No Arithmetic Exception ");
throw new ArithmeticException();
}
catch(ArithmeticException e)
{
System.out.println("Division By Zero in Arithmetic Exception");
}
a[0]=700;
System.out.println("No Array Index Exception ");
throw new ArrayIndexOutOfBoundsException();
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Array Index out of Bounds Exception");
```

```
}
}
}
```

We will get the following output:

No Arithmetic Exception

Division By Zero in Arithmetic Exception

No Array Index Exception

Array Index out of Bounds Exception

throws keyword

The method has a return type, name of the function and the arguments but we may also see a throws keyword appended at the end of the method for example:

```
int get_Data (int n) throws IOException
```

Now we understand the return type and arguments but the question is what this throws IOException is. This means that the function is dealing with the input/output operations in its definition and it has not taken care of exceptions that may occur while it gets invoked. So it's the responsibility of the programmer to take care of the exception mentioned in the throws statement while executing or calling this method. So if we have to call this method we have to write:

```
try

{

objname.get_Data(2);

}

catch(IOException obj)

{

}
```

In the following example we used the throwsNumberFormatException with

get_Data method. So in order to call this, we have to enclose it using try and catch blocks:

```
package book;

class MyCls

{

static int get_Data(String a) throws NumberFormatException

{

return(Integer.parseInt(a));

}

}


public class Throws_Demo {

public static void main(String[] args) {

int n;

try

{

n= MyCls.get_Data("200");

System.out.println("Data = " + n);

}

catch(NumberFormatException e)
```

{

}

}

}

We will get the following output

Data = 200

Understanding built-in exception classes

As discussed in the previous sections, it's important for a programmer to look out for the places in the program where the exception can take place and can accordingly handle those exceptions. As a programmer we must understand the various types of exceptions that may occur and what are the built-in exception classes, The following are some of the exception classes available in Java:

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incor
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined en
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting c
IllegalStateException	Environment or application is in incorrect s
IllegalThreadStateException	Requested operation not compatible with c
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a st
TypeNotPresentException	Type not found.

UnsupportedOperationException	An unsupported operation was encountered
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not im
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract c
InterruptedException	One thread has been interrupted by another
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

Table 6.1: Java's checked exceptions defined in java.lang

Creating user-defined exception

As all exceptions are inherited from the Exception class, we may create a user defined exception by extending the Exception class. As this is not the inbuilt exception, we must use the throw to forcibly throw the exception. The below example explains how we can create that exception and throw that exception.

The Exception class does not define any methods of its own. It does, of course, inherit those methods provided by Throwable. Thus, all exceptions, including those that you create, have the methods defined by Throwable available to them. They are shown in Table 6.2:

Method	Description
Throwable fillInStackTrace( )	Returns a Throwable objec
Throwable getCause( )	Returns the exception that
String getLocalizedMessage( )	Returns a localized descrip
String getMessage( )	Returns a description of the
StackTraceElement[ ] getStackTrace( )	Returns an array that conta
Throwable initCause(Throwable causeExc)	Associates causeExc with t
void printStackTrace( )	Displays the stack trace.
void printStackTrace(PrintWriter stream)	Sends the stack trace to the
void setStackTrace(StackTraceElement elements[ ])	Sets the stack trace to the e
String toString( )	Returns a String object con

Table 6.2: The methods defined by Throwable

In the following Java program, we created our own exception by extending the Exception class and forcibly throwing that exception by making use of throw keyword:

```
package book;

class MyOwnException extends Exception

{

MyOwnException(String msg)

{

super(msg);

}

}

public class Exception_User {

public static void main(String[] args) {

int a,b,c;

a=10;

b=10;

System.out.println("We are Trying to Create our Own Exceptions");

try
```

```
{
c= a-b;
throw new MyOwnException("Number is Equal to Zero");
}
catch(MyOwnException e)
{
System.out.println(e.getMessage());
}
}
}
```

We will get the following output:

We are Trying to Create our Own Exceptions

Number is Equal to Zero

Assignments

What are the advantages of exception handling? How it is different from a normal error?

What are the various types of exceptions? Explain with suitable examples?

What is the use of the finally block? How is it different from try and catch blocks?

What is the difference between throw and throw? Explain with suitable examples?

Define an exception called NameException that is thrown when the string is not equal to Hello Java? Write a Java program that uses this exception

Write a Java program that handles two nested exceptions while dealing with arrays?

CHAPTER 7

Multithreading in Java

Structure

Introduction

Creating a thread

Lifecycle of a thread

Thread priorities

Thread synchronization

Inter-thread communication

Assignments

Objective

One of the reason for the popularity of Java is multithreading. The Thread lifecycle is explained so as to write multi-threaded programs in Java. This chapter also illustrates the inter thread communication.

Introduction

Multithreading is one of the powerful tool of Java language that makes it different and efficient from other programming languages. Multithreading is a conceptual programming paradigm where a program is divided into two or more subprograms, which can be implemented at the same time in parallel. The program in operating system terminology is called as a process and as such multiple sub processes are executing simultaneously. The main advantage of multithreading is that the programmer can execute multiple things simultaneously for example one subprogram may display the animation on the screen and another subprogram may get the data from a large database. This multitasking allows the programmer to give an impression that everything is working simultaneously. Normally, a Java program executes from top to bottom for example the below program:

```
package book;

public class Mult_Threading {

public static void main(String[] args) {

System.out.println("Beginning of The Program");

for(int i=0;i<10;i++)

{

System.out.println("Body of The Program");

}

System.out.println("End of The Program");

}
```

}

The output of the above program is as expected:

Beginning of The Program

Body of The Program

Body of The Program

Body of The Program

Body of The Program

Body of The Program

Body of The Program

Body of The Program

Body of The Program

Body of The Program

Body of The Program

End of The Program

As in multithreading, the same program is divided into multiple thread, the output can be random as multiple processes will be executing simultaneously. It is important to understand that only a single processer is executing the multiple threads it does not mean they are actually getting executed at the same time but the executions are switched from one process to another which gives the impression that all are getting executed simultaneously. The switching of control is done by the Java interpreter. If we have a main method which behaves as a

main thread and three other threads then the switching is described as depicted in the diagram:

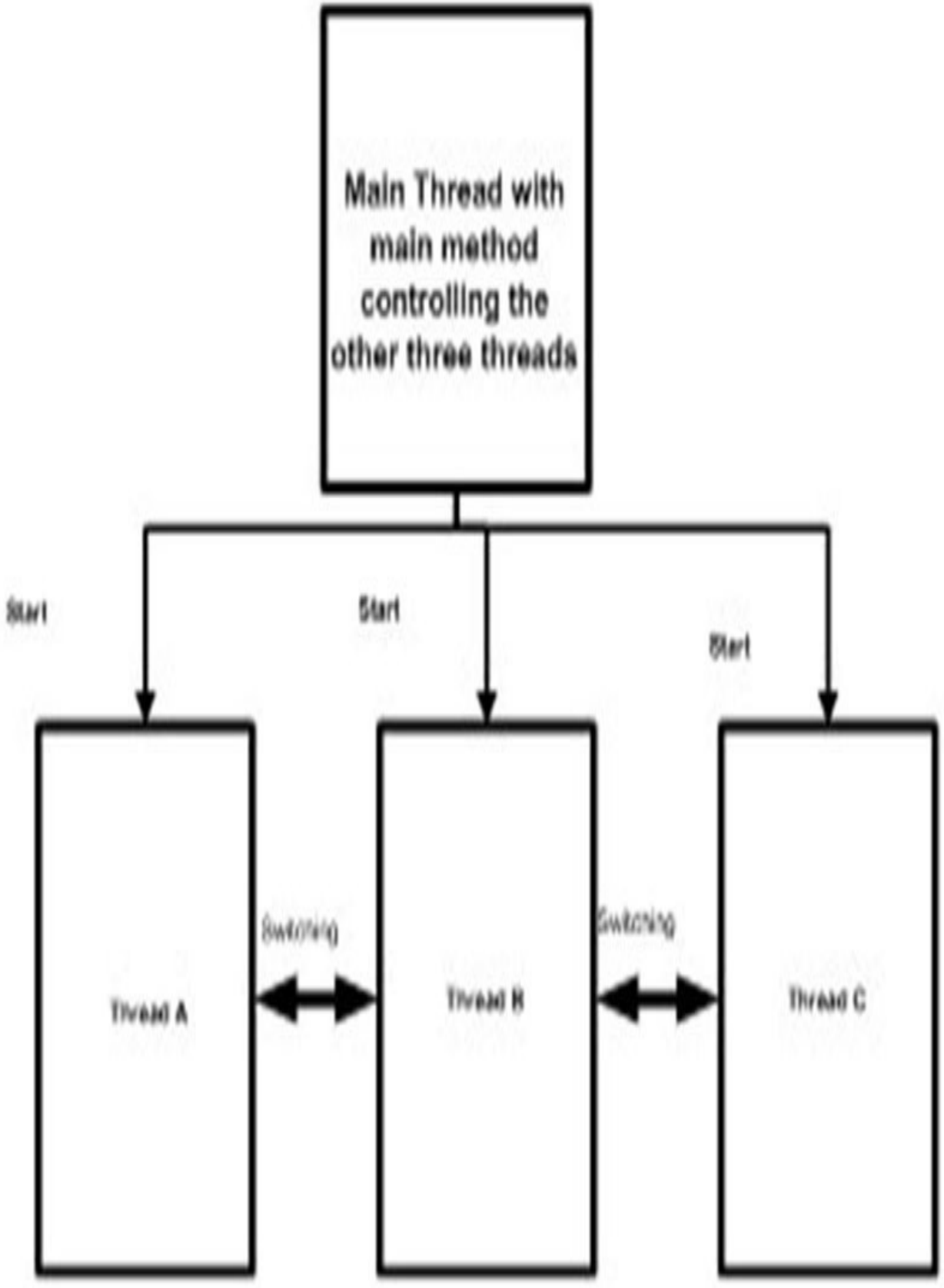

Main Thread with main method controlling the other three threads
Start
Start
Start
Thread A
Thread B
Thread C

Figure 7.1: Multithreading

Creating athread

The following two ways can be used to create threads in Java.

By extending the Thread class and overriding the run() method. If we want to implement any of the class as a thread, we must inherit the built-in Thread class. The Thread class is in java.lang package and we need not to import it as this is the default package. The Threadclass has a run() method and that method must be redefined in the derived class with all the code that we want to execute as a thread. So the instance of the program segment is as under:

```
class MyThread extends Thread

{

public void run()

{

for(int i=0;i<10;i++)

{

System.out.println(" From MyThread i = " + i);

}

}

}
```

Now we want to execute the above thread and that has to be done from the main thread as under:

```
MyThread ta = new MyThread();

ta.start();
```

After calling the start method of Thread class, it is not mandatory that the thread will start running as it may be busy with some other threads and it may put it in runnable state:

The Thread class has numerous methods which can be used with the objects of Thread class. For example if we want this thread to stop when the value of i = 2 then we can write:

```
if (i==2)

ta.stop();
```

The Thread class defines several methods that help manage threads. The ones that will be used in this chapter are shown here:

Method	Meaning
getName	Obtain a thread's name.
getPriority	Obtain a thread's priority.
isAlive	Determine if a thread is still running.

join	Wait for a thread to terminate.
run	Entry point for the thread.
sleep	Suspend a thread for a period of time.
start	Start a thread by calling its run method.

Table 7.1: Thread methods

The following examples illustrate the use of Thread class in implementing the multithreading.

In the below multithreading program three threads are created with three different for loops running in the run() method. All the three threads are started from the main thread that is the main method. After the execution of this program, we can see that all the four threads are executing simultaneously and the output cannot be predicted.

```
package threadingmca;

class A extends Thread

{

public void run()

{

for(int i= 1 ;i<10 ;i++)

{

System.out.println("From Thread A i =" + i);

}

}

}
```

```
class B extends Thread
{
public void run()
{
for(int j= 1 ;j<10 ;j++)
{
System.out.println("From Thread B j =" + j);
}
}
}
class C extends Thread
{
public void run()
{
for(int k= 1 ;k<10 ;k++)
{
System.out.println("From Thread C K =" + k);
}
}
}
public class ThreadingMca {
```

```
public static void main(String[] args) throws InterruptedException {
A threada = new A();
B threadb = new B();
C threadc = new C();
threada.start();
threadb.start();
threadc.start();
for(int j= 1 ;j<10 ;j++)
{
System.out.println("From Main =" + j);
}
System.out.println("All done");
}
}
```

We will get the following output:

From Thread A i =1

From Thread C K =1

From Thread B j =1

From Main =1

From Main =2

From Main =3

From Main =4

From Main =5

From Main =6

From Main =7

From Main =8

From Main =9

All done

From Thread B j =2

From Thread B j =3

From Thread B j =4

From Thread B j =5

From Thread B j =6

From Thread B j =7

From Thread B j =8

From Thread B j =9

From Thread C K =2

From Thread C K =3

From Thread C K =4

From Thread C K =5

From Thread C K =6

From Thread C K =7

From Thread C K =8

From Thread C K =9

From Thread A i =2

From Thread A i =3

From Thread A i =4

From Thread A i =5

From Thread A i =6

From Thread A i =7

From Thread A i =8

From Thread A i =9

In the following multithreading program, different methods of Thread class are used:

```
package threadingmca;

class A extends Thread

{

public void run()

{
```

```
for(int i= 1 ;i<10 ;i++)
{
if (i==2)
stop();
System.out.println("From Thread A i =" + i);
}
}
}
class B extends Thread
{
public void run()
{
for(int j= 1 ;j<10 ;j++)
{
if (j==2)
try {
sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println("From Thread B j =" + j);
```

```
}
}
}
class C extends Thread
{
public void run()
{
for(int k= 1 ;k<10 ;k++)
{
System.out.println("From Thread C K =" + k);
}
}
}
public class ThreadingMca {
public static void main(String[] args) throws InterruptedException {
A threada = new A();
B threadb = new B();
C threadc = new C();
threada.start();
threadb.start();
threadc.start();
```

```
for(int j= 1 ;j<10 ;j++)
{
System.out.println("From Main =" + j);
}
System.out.println("All done");
}
}
```

We will get the following output:

From Main =1

From Thread A i =1

From Thread C K =1

From Thread C K =2

From Thread C K =3

From Thread C K =4

From Thread C K =5

From Thread C K =6

From Thread C K =7

From Thread C K =8

From Thread C K =9

From Main =2

From Main =3

From Main =4

From Main =5

From Main =6

From Main =7

From Main =8

From Main =9

All done

From Thread B j =1

From Thread B j =2

From Thread B j =3

From Thread B j =4

From Thread B j =5

From Thread B j =6

From Thread B j =7

From Thread B j =8

From Thread B j =9

The Runnable interface

The other method to implement multithreading is by implementing the Runnable interface. This Runnable interface has a run() method which must be implemented by the class implementing the Runnable interface. All the code that we want to be executed as a thread must be kept in this run() method:

```
class MyClass implements Runnable

{

// Code to be executed as thread

}
```

Now, after implementing the Runnable create a Thread object that is instantiated from the above Runnable class as under:

```
MyClass r = new MyClass();

Thread ta = new Thread(r);
```

Now after creating the thread type object, start the thread by calling the start method of Thread class as under:

```
ta.start();
```

The below examples illustrates the implementation Runnable interface is implemented by three different classes and run method is implemented. In the main method The Thread classes are created which are instantiated from these Runnable interface classes:

```
package threadingmca;

class A implements Runnable

{

public void run()

{

for(int i= 1 ;i<10 ;i++)

{

System.out.println("From Thread A i =" + i);

}

}

}

class B implements Runnable

{

public void run()

{

for(int j= 1 ;j<10 ;j++)
```

```
{
System.out.println("From Thread B j =" + j);
}
}
}
class C implements Runnable
{
public void run()
{
for(int k= 1 ;k<10 ;k++)
{
System.out.println("From Thread C K =" + k);
}
}
}

public class ThreadingMca {
public static void main(String[] args) throws InterruptedException {
A runnablea = new A();
B runnableb = new B();
C runnablec = new C();
```

```
Thread ta = new Thread(runnablea);
Thread tb = new Thread(runnableb);
Thread tc = new Thread(runnablec);
ta.start();
tb.start();
tc.start();
for(int j= 1 ;j<10 ;j++)
{
System.out.println("From Main =" + j);
}
System.out.println("All done");
}
}
```

We will get the following output:

From Thread B j =1

From Thread C K =1

From Thread C K =2

From Main =1

From Main =2

From Main =3

From Main =4

From Main =5

From Main =6

From Main =7

From Main =8

From Main =9

All done

From Thread C K =3

From Thread C K =4

From Thread C K =5

From Thread C K =6

From Thread C K =7

From Thread C K =8

From Thread C K =9

From Thread B j =2

From Thread B j =3

From Thread B j =4

From Thread B j =5

From Thread B j =6

From Thread B j =7

From Thread B j =8

From Thread B j =9

From Thread A i =1

From Thread A i =2

From Thread A i =3

From Thread A i =4

From Thread A i =5

From Thread A i =6

From Thread A i =7

From Thread A i =8

From Thread A i =9

Lifecycle of a thread

It is important to understand the life cycle of a thread. The various states of a thread are:

New born state: This is when the object of the new thread is created. It will not go directly to the active state until the start method is called.

Running and runnable state: A thread goes into either of the two states when the start method is called. As the processor can execute only one thread at a time, only one thread is getting executed and other thread is in runnable state.

Blocked state: If we want to suspend one of the thread for some duration of time we can use methods like sleep, wait and suspend. After the specified time interval the notify or resume method is called to take the thread into the active state.

Dead state: If at any point of time we want to kill the thread, we can call the stop method and the thread will go into the dead state.

The following diagram gives the full description about the lifecycle of a Thread:

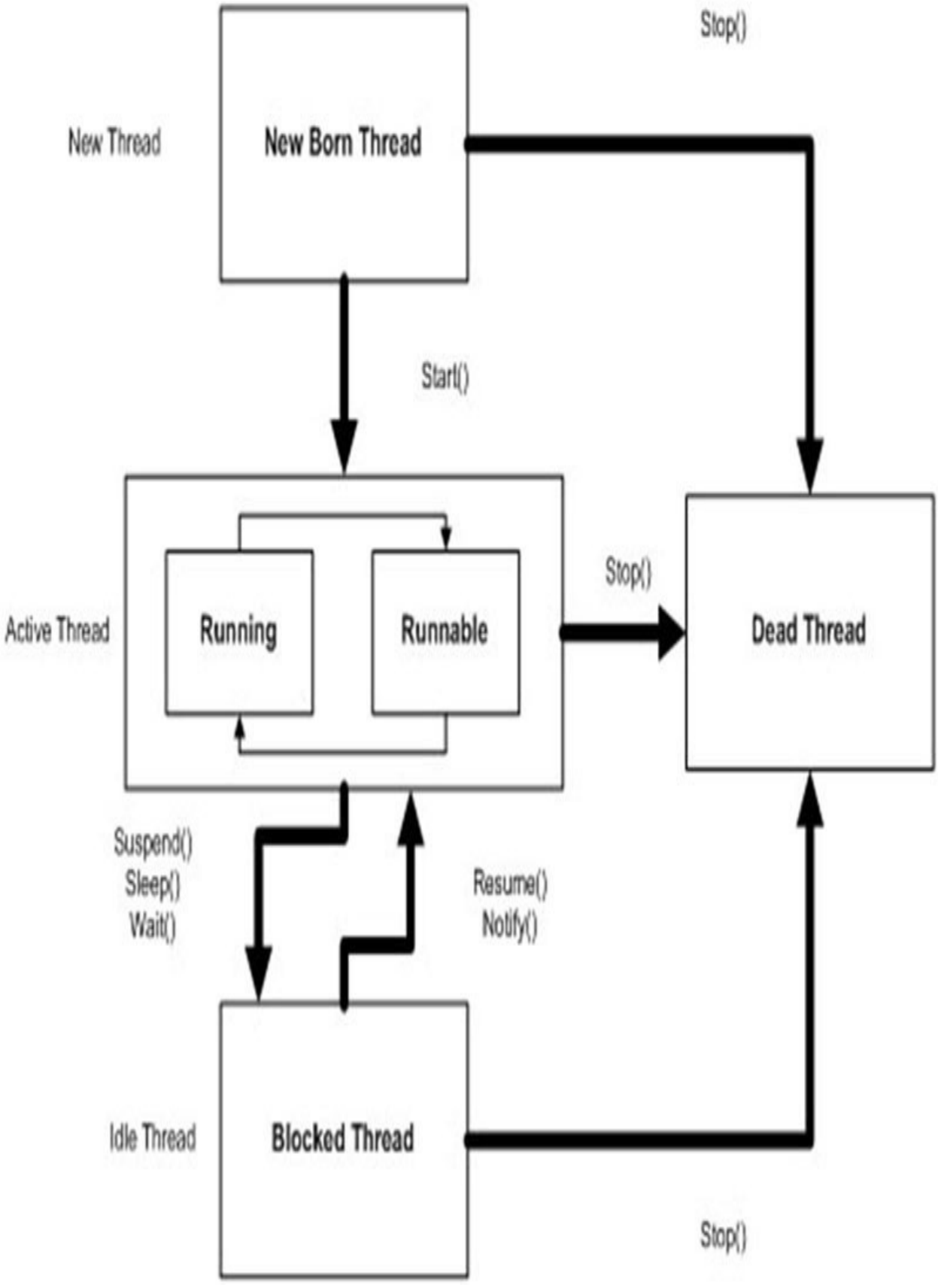

Stop()
New Thread
New Born Thread
Start()
Active Thread
Running
Runnable
Stop()
Dead Thread
Suspend()
Sleep()
Wait()
Resume()
Notify()
Idle Thread
Blocked Thread
Stop()

Figure 7.2: Lifecycle of thread

The below Java program overrides the start method of the thread class and displays the sequence in which different threads are getting executed. The other methods in Thread class are declared as final and hence cannot be overridden:

```
package threadingmca;

class A extends Thread

{

public void run()

{

for(int i= 1 ;i<5 ;i++)

{

if (i==2)

stop();

System.out.println("From Thread A i =" + i);

}

}

public void start()

{

super.start();
```

```
System.out.println("Thread A Started" );
}
}
class B extends Thread
{
public void run()
{
for(int j= 1 ;j<5 ;j++)
{
if (j==2)
try {
sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println("From Thread B j =" + j);
}
}
public void start()
{
super.start();
```

```
System.out.println("Thread B Started" );
}
}
class C extends Thread
{
public void run()
{
for(int k= 1 ;k<5 ;k++)
{
System.out.println("From Thread C K =" + k);
}
}
public void start()
{
super.start();
System.out.println("Thread C Started" );
}
}
public class ThreadingMca {
public static void main(String[] args) throws InterruptedException {
A threada = new A();
```

```
B threadb = new B();
C threadc = new C();
threada.start();
threadb.start();
threadc.start();
for(int j= 1 ;j<5 ;j++)
{
System.out.println("From Main =" + j);
}
System.out.println("All done");
}
}
```

We will get the following output:

Thread A Started

From Thread A i =1

Thread B Started

From Thread B j =1

Thread C Started

From Main =1

From Thread C K =1

From Main =2

From Thread C K =2

From Main =3

From Thread C K =3

From Main =4

All done

From Thread C K =4

From Thread B j =2

From Thread B j =3

From Thread B j =4

Thread priorities

In multithreading, each thread is given a priority that affects the way Java scheduler is giving priority to any of the threads. In order to set the priority of a thread the following method is used:

ThreadName.setPriority(int value);

The value can take any of the constants as defined in Thread class as under:

MIN_PRIORITY = 1

NORM_PRIORITY = 5

MAX_PRIORITY =10

For example if we want to set the highest priority of a threadA, we can use the following statement:

threadA.setPriority(10);

The following Java program creates another example of multithreading by setting the priorities of threads:

```
package threadingmca;
class A extends Thread
{
public void run()
{
for(int i= 1 ;i<5 ;i++)
{
System.out.println("From Thread A i =" + i);
}
}

public void start()
{
super.start();
System.out.println("Thread A Started" );
}
}
class B extends Thread
{
public void run()
{
```

```
for(int j= 1 ;j<5 ;j++)
{
if (j==2)
try {
sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
System.out.println("From Thread B j =" + j);
}
}
public void start()
{
super.start();
System.out.println("Thread B Started" );
}
}
class C extends Thread
{
public void run()
{
```

```
for(int k= 1 ;k<5 ;k++)
{
System.out.println("From Thread C K =" + k);
}
}
public void start()
{
super.start();
System.out.println("Thread C Started" );
}
}
public class ThreadingMca {
public static void main(String[] args) throws InterruptedException {
A threada = new A();
B threadb = new B();
C threadc = new C();
threada.setPriority(10);
threada.start();
threadb.start();
threadc.start();
for(int j= 1 ;j<5 ;j++)
```

```
{
System.out.println("From Main =" + j);
}
System.out.println("All done");
}
}
```

We will get the following output:

Thread A Started

From Thread A i =1

From Thread A i =2

Thread B Started

From Thread A i =3

From Thread A i =4

From Thread B j =1

Thread C Started

From Main =1

From Main =2

From Thread C K =1

From Main =3

From Main =4

All done

From Thread C K =2

From Thread C K =3

From Thread C K =4

From Thread B j =2

From Thread B j =3

From Thread B j =4

Thread synchronization

So far we have put the code within the run method of each thread. What if the threads are accessing the common resource outside their run method for example a same file is getting accessed by one thread reading the file and another thread writing on the same file. This can lead to a serious problems. Java comes up with the solution of such common resources by creating that portion of the code as synchronized as:

```
synchronized void commonupdate

{

..// code that needs to be synchronized

..

}
```

The following Java program explains the use of synchronized block. In this program, two threads are trying to access the same block of code. The block is kept as synchronized:

```
package threadingdemo;

class counter

{

int count;
```

```
public synchronized void increment()
{
count++;
}
}
public class mc {
public static void main(String[] args)throws Exception {
counter obj = new counter();
Thread ta = new Thread( new Runnable() {
@Override
public void run() {
for(int i=0;i<1000;i++)
obj.increment();
}
});
Thread tb = new Thread(new Runnable() {
@Override
public void run() {
for(int j=0;j<1000;j++)
obj.increment();
}
```

```
});
ta.start();
tb.start();
ta.join();
tb.join();
System.out.println("Count =" + obj.count);
System.out.println("End of Program");
}
}
```

We will get the following output:

Count =2000

End of Program

In the belowJava program the mixed-up output of the three message strings is produced. As nothing exists to stop all three threads from calling the same method, on the same object, at the same time. This is known as a race condition, because the three threads are racing each other to complete the method. The same program is repeated with the synchronized keyword and the correct output is produced:

```
package book;
```

```
class Action_Class {
void call(String content)
{
System.out.print("start " + content);
try {
Thread.sleep(1000);
} catch(InterruptedException e) {
System.out.println("Exception");
}
System.out.println(" end");
}
}

class Trigger implements Runnable {
String content;
Action_Class target;
Thread t;
public Trigger(Action_Class targ, String s)
{
target = targ;
content = s;
```

```
t = new Thread(this);
t.start();
}
public void run() {
target.call(content);
}
}

public class SyncDemo {
public static void main(String args[]) {
Action_Class target = new Action_Class();
Trigger ob1 = new Trigger(target, "NIELIT");
Trigger ob2 = new Trigger(target, "SRINAGAR");
Trigger ob3 = new Trigger(target, "J&K ");
try {
ob1.t.join();
ob2.t.join();
ob3.t.join();
} catch(InterruptedException e){
System.out.println("Interrupted");
}
```

```
}
}
```

We will get the following output:

start SRINAGARstart NIELITstart J&K end

end

end

The above same program with synchronized block:

```
package book;
class Action_Class {
synchronized void call(String content)
{
System.out.print("start " + content);
try {
Thread.sleep(1000);
} catch(InterruptedException e){
System.out.println("Exception");
}
```

```
System.out.println(" end");
}
}

class Trigger implements Runnable {
String content;
Action_Class target;
Thread t;
public Trigger(Action_Class targ, String s)
{
target = targ;
content = s;
t = new Thread(this);
t.start();
}
public void run() {
target.call(content);
}
}

public class SyncDemo {
```

```
public static void main(String args[]) {
Action_Class target = new Action_Class();
Trigger ob1 = new Trigger(target, "NIELIT");
Trigger ob2 = new Trigger(target, "SRINAGAR");
Trigger ob3 = new Trigger(target, "J&K ");
try {
ob1.t.join();
ob2.t.join();
ob3.t.join();
} catch(InterruptedException e){
System.out.println("Interrupted");
}
}
}
```

We will get the following output:

start NIELIT end

start J&K end

start SRINAGAR end

Inter-thread communication

We can achieve the synchronization of threads through a more subtle level of control through inter-process communication. To stay away from polling, Java includes awell-designed interprocess communication method via thewait(), notify(), and notifyAll() methods. These methods are implemented as final methods in Object, so all classes automatically have them. All three methods can be called only from within a synchronized context:

wait( ) tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).

notify( ) wakes up a thread that called wait( ) on the same object.

notifyAll( ) wakes up all the threads that called wait( ) on the same object. One of the threads will be granted access.

These methods are declared within Object, as shown here:

final void wait( ) throws InterruptedException

final void notify( )

final void notifyAll( )

In the following Java program, the synchronized block is implemented but still we are not able to get the desired output as the question 0 should get the answer 0 and so on. In the next program, we implemented the inter process communication to address this issue:

```
package book;

class Ques {

int n;

synchronized int get()

{

System.out.println("Answer " + n);

return n;

}

synchronized void put(int n)

{

this.n = n;

System.out.println("Question " + n);

}

}


class Teacher implements Runnable

{
```

```
Ques Ques;
Teacher(Ques Ques) {
this.Ques = Ques;
new Thread(this, "Teacher").start();
}
public void run() {
int i = 0;
while(i<5)
{
Ques.put(i++);
}
}
}

class Student implements Runnable {
Ques Ques;
Student(Ques Ques) {
this.Ques = Ques;
new Thread(this, "Student").start();
}
public void run() {
```

```
int j=0;
while(j<5) {
Ques.get();
j++;
}
}
}

public class ThreadCommunication {
public static void main(String args[]) {
Ques Ques = new Ques();
new Teacher(Ques);
new Student(Ques);
}
}
```

We will get the following output:

Question 0

Question 1

Question 2

Question 3

Question 4

Answer 4

Answer 4

Answer 4

Answer 4

Answer 4

The same program is implemented through interprocess communication. The wait(), notify (), and notifyAll() methods are used to get the inter process communication and to achieve the correct results:

```
package book;

class Ques1 {

int n;

boolean valueSet = false;

synchronized int get() {

while(!valueSet)

try {

wait();

} catch(InterruptedException e)

{ System.out.println("InterruptedException caught");
```

```
}
System.out.println("Answer " + n);
valueSet = false;
notify();
return n;
}

synchronized void put(int n)
{
while(valueSet)
try {
wait();
} catch(InterruptedException e)
{ System.out.println("InterruptedException caught");
}
this.n = n;
valueSet = true;
System.out.println("Question" + n);
notify();
}
}
```

```
class Teacher1 implements Runnable {
Ques1 Ques1;
Teacher1(Ques1Ques1) {
this.Ques1 =Ques1;
new Thread(this, "Teacher1").start();
}
public void run() {
int i = 0;
while(i<5) {
Ques1.put(i++);
}
}
}

class Student1 implements Runnable {
Ques1 Ques1;
Student1(Ques1 Ques1) {
this.Ques1 = Ques1;
new Thread(this, "Student1").start();
}
```

```
public void run() {
int j=0;
while(j<5) {
Ques1.get();
}
}
}

public class InterProcessComm {
public static void main(String args[])
{
Ques1 Ques1 = new Ques1();
new Teacher1(Ques1);
new Student1(Ques1);
}
}
```

We will get the following output:

Question0

Answer 0

Question1

Answer 1

Question2

Answer 2

Question3

Answer 3

Question4

Answer 4

Assignments

What is the advantage of multithreading? How this enhances the efficiency of a Java program?

Explain the complete lifecycle of a thread?

What are the different ways of implementing multithreading in Java? Explain with relevant examples?

What are the various methods defined in Thread class, which can be used to suspend, start and stop a thread?

Write a Java program that explains the use of synchronization in multithreading?

CHAPTER 8

Applets in Java

Introduction

Java uses applets primarily for internet computing. Applets are small Java programs which can be executed by the applet viewer or any of the browsers that supports applets. If we want to take input, perform some arithmetic operations, play sounds, display graphics, create animations, and play any of the interactive games through the html pages, we can make use of an applet for wide range of Java applications.

Structure

Introduction

Features of applets

Understanding an Applet class

Applet lifecycle

Understanding the applet tag

Creating a web page with applet

Passing parameters to applets

Handling applet events

Assignments

Objective

This chapter explains Applets in detail. After reading this chapter, we should be able to create applets and handle applet events.

Features of applets

To understand the working of a basic applet, let us create an applet with the following code:

```
import java.applet.*;

import java.awt.*;

public class MyApplet extends Applet

{

public void paint(Graphics g)

{

g.drawString(" Welcome to Applets", 20, 20);

}

}
```

The above program creates a simple applet that displays Welcome to Applets.

The following points are to be kept in mind while using the applets:

Applets do not have a main method. Applets have a life cycle (To be discussed)

wherein there execution starts from the init () method.

To create an applet, we must import the java.applet package as it contains the Applet class.Applet class provides all required support for applet implementation, such as starting and stopping an applet. It moreover provides methods that load and display images, and methods that load and play audio clips.

The applet class must be kept as public as it is accessible from outside the class. In the above case we have created MyApplet class with public visibility and the paint method with the public access specifier.

To run an applet, the applet tag can be used as under:

/* <applet code=MyApplet Height=200 Width=200>

*/

This tag must be enclosed in a multi-line comment. Now to execute an applet, the applet viewer can be used as:

appletviewer MyApplet.java

To execute an applet from within the HTML file, the same tag can be inserted within the HTML file as:

<html>

<body>

/* <applet code=MyApplet Height=200 Width=200>

*/

</body>

</html>

The paint method can be used to display the string of messages. The Graphics type object is used to run various methods in Graphics class defined in java.awt package. Output to applet's window is not performed by System.out.println( ). Rather it is handled with various AWT methods, such as drawString(), which outputs a string to a specified X,Y position. Input is also handled in a different way than in a console application.

The Applet class imported from the java.applet package defines the methods shown in the table below. Applet extends the AWT class Panel. In turn, Panel extends Container, which extends Component. These classes provide support for Java's window-based, graphical interface:

Method	Description
void destroy( )	Called by the browser just
AccessibleContextget AccessibleContext( )	Returns the accessibility c
AppletContext getAppletContext( )	Returns the context associ
String getAppletInfo( )	Returns a string that descri
AudioClip getAudioClip(URL url)	Returns an AudioClip obje
AudioClip getAudioClip(URL url, String clipName)	Returns an AudioClip obje
URL getCodeBase( )	Returns the URL associate

URL getDocumentBase( )	Returns the URL of the HT
Image getImage(URL url)	Returns an Image object th
Image getImage(URL url)	Returns an Image object th
Image getImage(URL url, String imageName)	Returns an Image object th
Locale getLocale( )	Returns a Locale object tha
String getParameter(String paramName)	Returns the parameter assc
String[][] getParameterInfo( )	Returns a String table that
void init( )	Called when an applet beg
boolean isActive( )	Returns true if the applet h
static final AudioClipnew AudioClip(URL url)	Returns an AudioClip obje
newAudioClip (URL url)	Audio clip found at the loc
void play(URL url)	If an audio clip is found at
void play(URL url, String clipName)	If an audio clip is found at
void resize(Dimension dim)	Resizes the applet accordir
void resize(int width, int height)	Resizes the applet accordir
final void setStub(AppletStub stubObj)	Makes stubObj the stub for
void showStatus(String str)	Displays str in the status w
void start( )	Called by the browser whe
void stop( )	Called by the browser to su

Table 8.1: Methods defined by Applet class

Creating an Applet

Every applet created is a subclasses of Applet class. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. Execution of an applet does not start at main( ). In fact, few applets even have main( ) methods. As an alternative, implementation of an applet is started and controlled with a completely different method, which will be explained in this chapter.

This simple applet is created using a Notepad. The init() method is used for initialization and the paint() method is used to display a simple message. The program is executed using the applet viewer:

```
/*<applet code="MyFirstApplet" width="200" height="400">

</applet>

*/

import java.applet.Applet;

import java.awt.*;

public class MyFirstApplet extends Applet

{

public void init()

{

}
```

```
public void paint(Graphics g)

{

g.drawString("Welcome to Applets" 100, 200);

}

}
```

To run an applet, it is specified in HTML file. One way to accomplish this is by using the applet tag. The applet will be executed by a Java-enabled web browser when it encounters the applet tag within the HTML file. To view and test an applet more conveniently, simply include a comment at the head of your Java source code file that contains the applet tag. This way, our code is documented with the necessary HTML statements needed by our applet, and you can test the compiled applet by starting the applet viewer with our Java source code file specified as the target. Here is an example of such a comment:

```
/*

<applet code="MyFirstApplet" width="200" height="400">

</applet>

*/
```

This comment contains an applet tag that will run an applet called MyFirstApplet in a window that is 200 pixels wide and 200 pixels high. As the insertion of an applet command makes testing applets easier. The following output is generated using the applet viewer:

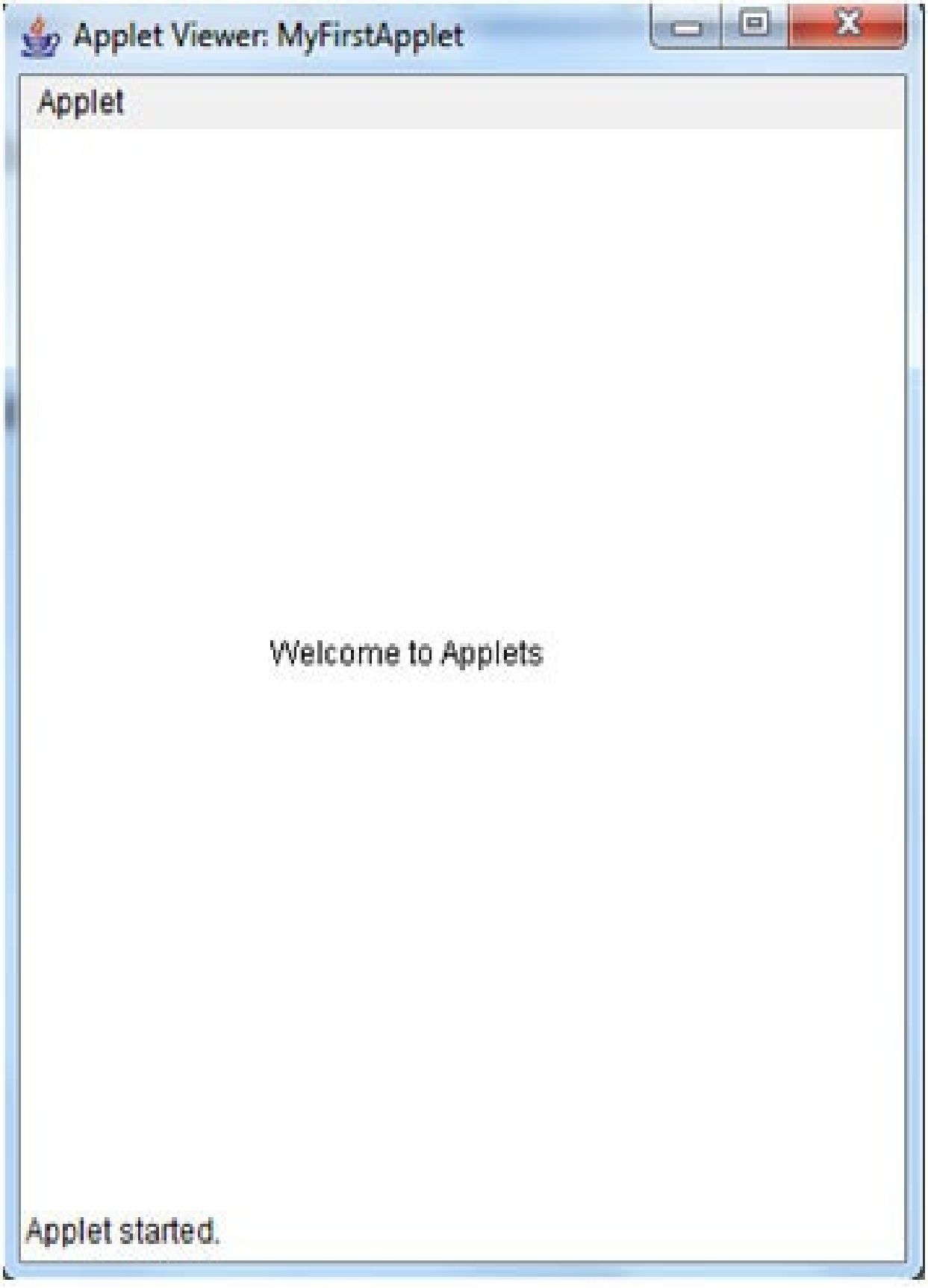

Applet Viewer: MyFirstApplet
Applet
Welcome to Applets
Applet started.

Figure 8.1: A running Applet using AppletViewer

A simple Java applet that sets the foreground and background colors and outputs a string:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="AppletExample2" width=500 height=100>
</applet>
*/
public class AppletExample2 extends Applet{
String msg;
// set the foreground and background colors.
public void init() {
setBackground(Color.cyan);
setForeground(Color.red);
msg = "init( ) Method Invoked";
}
// Initialize the string to be displayed.
```

```
public void start() {
msg += " start( ) Method Invoked";
}
// Display msg in applet window.
public void paint(Graphics g) {
msg += " paint( ) Method Invoked";
g.drawString(msg, 10, 30);
}
}
```

We will get the following output:


Applet Viewer: AppletExample2
Applet
init( ) Method Invoked start( ) Method Invoked paint( ) Method Invoked
Applet started.

Figure 8.2: Applet with various methods

A simple moving message applet is created This applet creates a thread that scrolls the message contained in msg right to left across the applet's window:

```
package book;

// A simple banner to display a moving message.

import java.awt.*;

import java.applet.*;

/*

<applet code="AppletBanner" width=300 height=50>

</applet>

*/

public class AppletBanner extends Applet implements Runnable

{

String msg = " A Simple Moving Message.";

Thread t = null;

int state;

boolean stopFlag;

//Set colors and initialize thread.
```

```
public void init() {
setBackground(Color.BLUE);
setForeground(Color.BLACK);
}
//Start thread
public void start() {
t = new Thread(this);
stopFlag = false;
t.start();
}
//Entry point for the thread that runs the banner.
public void run() {
char ch;
//Display banner
for( ; ; ) {
try {
repaint(); Thread.sleep(250);
ch = msg.charAt(0);
msg = msg.substring(1, msg.length());
msg += ch;
if(stopFlag)
```

```
break;
} catch(InterruptedException e) {}
}
}
//Pause the banner.
public void stop() {
stopFlag = true;
t = null;
}

//Display the banner.
public void paint(Graphics g) {
g.drawString(msg, 50, 30);
}
}
```

We will get the following output:

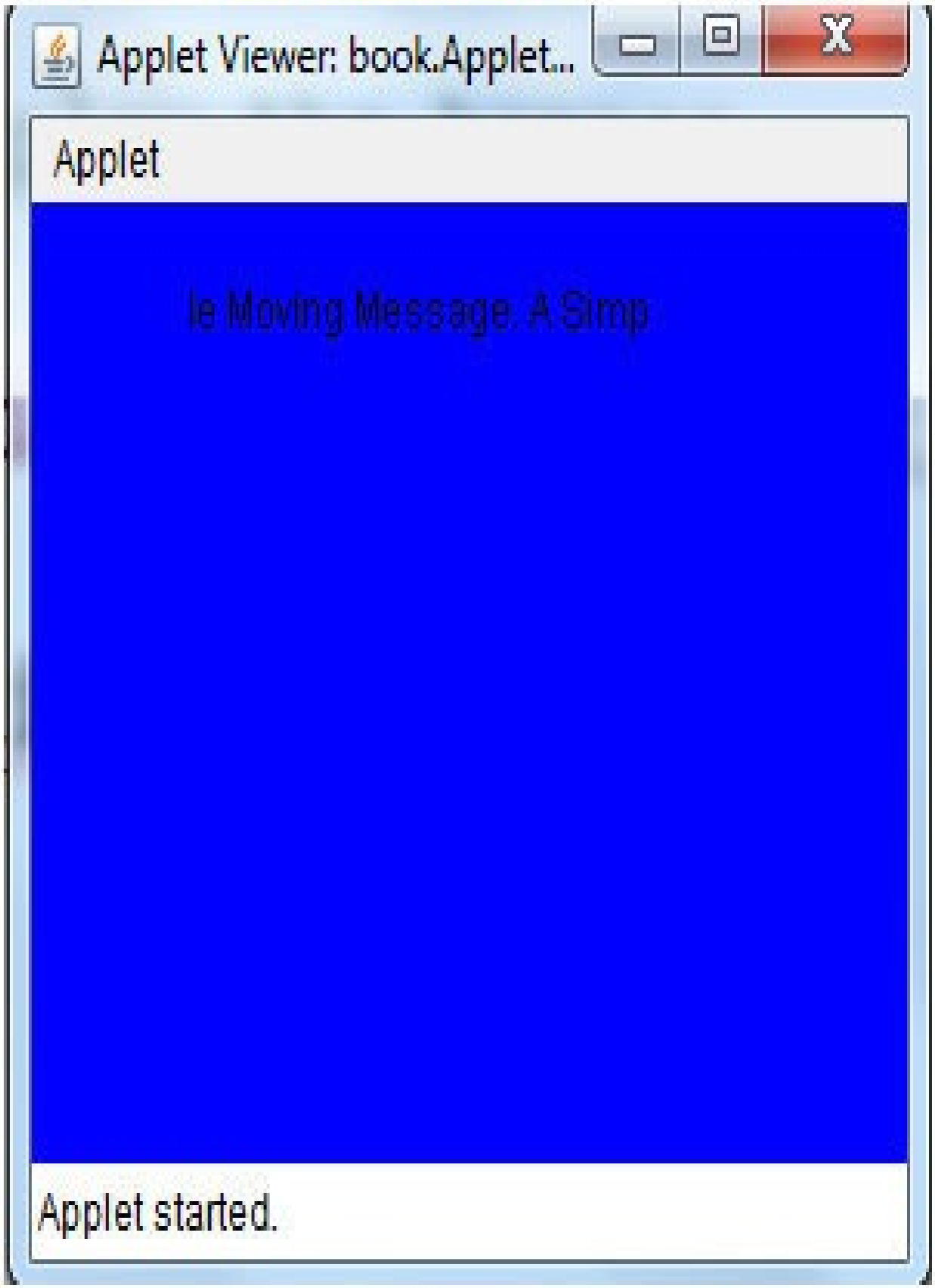

Applet Viewer: book.Applet...
Applet
le Moving Message: A Simp
Applet started.

Figure 8.3: Applet with a moving message

This Java program creates an applet by extending the JApplet. The different layouts are demonstrated in this applet:

```
package book;

import java.awt.*;

import javax.swing.*;

public class MyApplet extends JApplet{

Container c;

public void init()

{

c= getContentPane();

border();

}

public void border()

{

c.setLayout(new BorderLayout());

c.add(new Button("North Button"),BorderLayout.NORTH);

c.add(new Button("South Button"),BorderLayout.SOUTH);
```

```
c.add(new Button("East Button"),BorderLayout.EAST);
c.add(new Button("West Button"),BorderLayout.WEST);
c.add(new Button("Center Button"),BorderLayout.CENTER);
}
}
```

We will get the following output:


North Button
West Button
Center Button
East Button
South Button

Figure 8.4: BorderLayout example

Applet lifecycle

As we understood, how to create a simple applet inherited from the Applet class. It inherits some methods from the base class. In order to understand the working of an applet and the set of methods we must understand the lifecycle of an applet. Every Java applet goes through the states as described in the diagram. The various methods called during various states of an applet are as under.

The init( ) method is the initial method to be called. This is where applet initializes its variables. This method is called just the once throughout the runtime of an applet.

The start() method is called after init(). It is also called to resume an applet once it has been stopped. While init() is called once, the start() is called every time an applet's HTML page is displayed onscreen. Therefore, if a user leaves a webpage and arrives back, the applet resumes execution at start().

The paint() method is called every time an applet's output should be redrawn.

The stop() method is called as soon as a web browser leaves the HTML page containing the applet.

The destroy() method is called as soon as the settings conclude that the applet needs to be detached entirely from memory. The stop() method is always called prior to destroy():

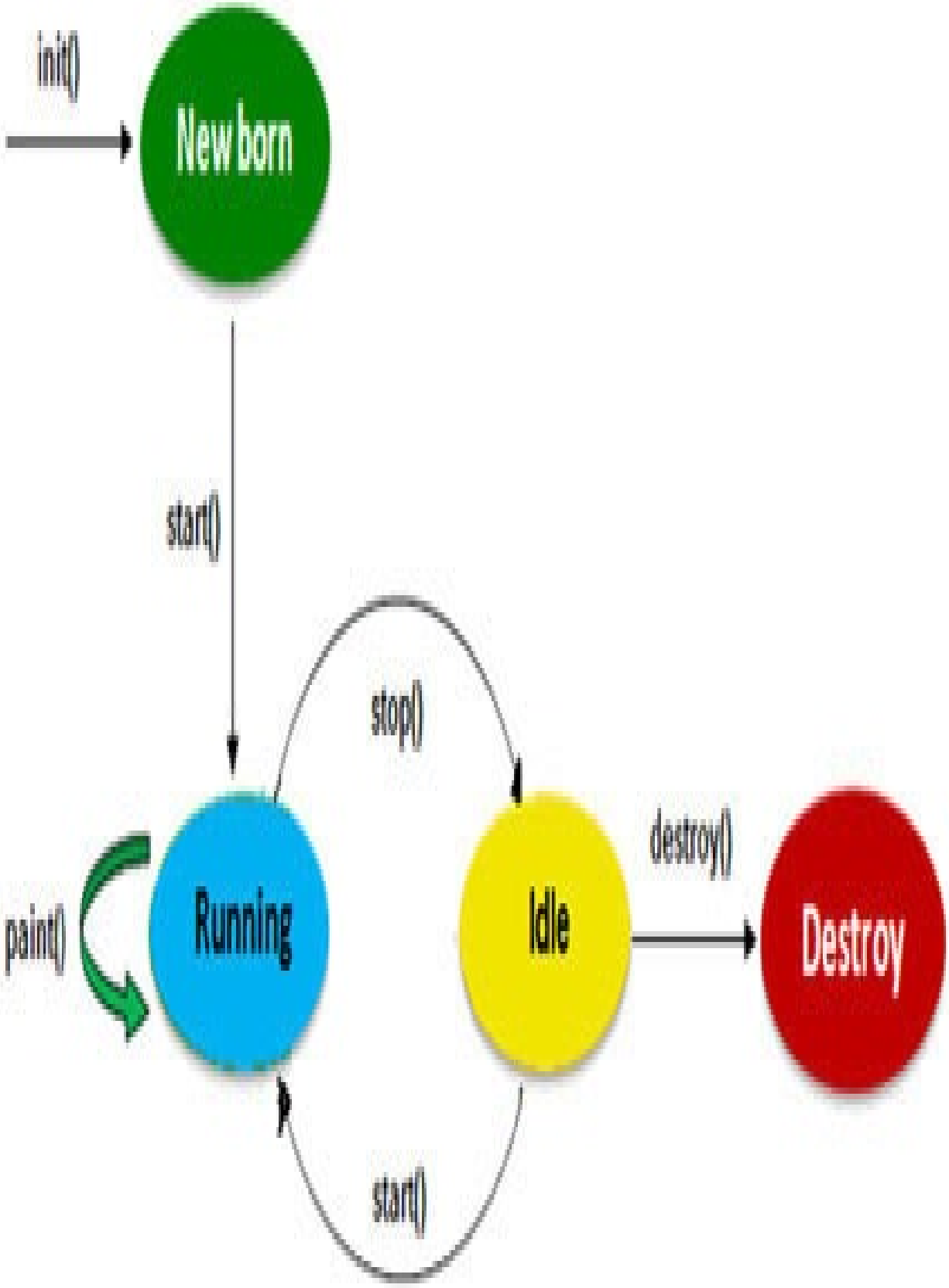

init()
Newborn
start()
stop()
paint()
Running
Idle
destroy()
Destroy
start()

Figure 8.5: Lifecycle of an Applet

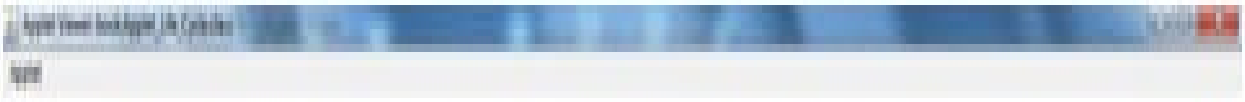
In the following Java program an applet is created to demonstrate the lifecycle of an applet. The sequence in which various methods are getting executed is displayed:

```
package book;

import java.applet.Applet;

import java.awt.Graphics;

import java.awt.Label;

public class Applet_Life_Cycle extends Applet {

String msg="";

@Override

public void destroy() {

super.destroy();

msg=msg+" Applet Destroyed ";

}


@Override

public void init() {

Label l = new Label("Life Cycle of Applet");
```

```
add(l);
super.init();
msg=msg+" Applet Initialized ";
}

@Override
public void start() {

super.start();
msg=msg+" Applet Started ";
}

@Override
public void stop() {

super.stop();
msg=msg+" Applet Stopped ";
}

public void paint(Graphics g)
{
```

```
msg=msg+" Paint Method ";
g.drawString(msg, 100, 100);
}
}
```

We will get the following output:



Life Cycle of Apple Life Cycle of Apple

Apple Initialized Apple Started Print Method Print Method Print Method Apple Stopped Apple Continued Apple Initialized Apple Started Print Method Print Method

Figure 8.6: Applet lifecycle

Understanding the Applet tag

So far we have used the applets with only the mandatory attributes, that is, code, width, and height. The applet tag have some other optional attributes which can be utilized to enhance the functionality. As already stated that the applet tag can be used within the Java program with multi-line comments and can be executed by the applet viewer. We can also use multiple applet tags from within the HTML file. In both the cases the applet tag can be used with full functionality. The applet tag with mandatory and optional attributes is as under:

< APPLET

[CODEBASE = code_baseURL]

CODE = applet_File

[ALT = alternate_Text]

[NAME = applet_Instance_Name]

WIDTH = Inpixels HEIGHT = Inpixels

[ALIGN = alignment]

[VSPACE = Inpixels]

[HSPACE = Inpixels]

[< PARAM NAME = Attribute_Name VALUE = Attribute_Value>]

[< PARAM NAME = Attribute_Name2 VALUE = Attribute_Value>] …

[HTML Displayed in the absence of Java]

</APPLET>

Each part of the applettag is explained as under:

[CODEBASE = code_baseURL]	Specif
CODE = applet_File	A man
[ALT = alternate_Text]	The A
[NAME = applet_Instance_Name]	An opt
WIDTH = Inpixels HEIGHT = Inpixels	Manda
[ALIGN = alignment]	An opt
[VSPACE = Inpixels]	This at
[HSPACE = Inpixels]	This at
[< PARAM NAME = Attribute_Name VALUE = Attribute_Value>]	The PA
[< PARAM NAME = Attribute_Name2 VALUE = Attribute_Value>] …	We can

Table 8.2: The applet tag, with mandatory and optional attributes

Passing parameters to Applets

The applet tag as already discussed, can pass parameters to the Java file. We can pass multiple parameters using the param tag and specifying the name and value which can further be processed in the Java file. The param tag has the following form:

< PARAM NAME = Attribute_Name1 VALUE = Attribute_Value1>

< PARAM NAME = Attribute_Name2 VALUE = Attribute_Value2>

For example, if we want to pass the sname and marks to the Java code, the following param tag is included within the applet tag.

<applet code="ParameterPassing" width=300 height=300>

<param name=sname value=Mubashir>

<param name=marks value=300>

</applet >

Now in order to retrieve the values in the Java program the getParameter method is used as under:

String Name = getParameter("sname");

String param = getParameter("marks");

As the second parameter is of type integer, the Integer class is used to covert the String into int as.

int mks = Integer.parseInt(param);

Thus, we are able to get the parameters from the applet tag of the HTML file. These data values can be further processed from within the program to generate the output.

The following program illustrates the parameter passing in applets. Here, an applet tag is created which passes four arguments to the applet code. The arguments are processed and displayed using the paint method:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="ParameterPassing" width=300 height=80>
<param name=CourseName value=Java>
<param name=marks_1 value=560>
<param name=Result value=Pass>
<param name=next_class value=true>
```

```
</applet >
*/
public class ParameterPassing extends Applet{
String CourseName="nnn";
int marks_1;
String Result;
boolean active;
public void start()
{
super.start();
String param;
CourseName = getParameter("CourseName");
if(CourseName == null)
CourseName = "Not Found";
param = getParameter("marks_1");
try {
if(param != null) // if not found
marks_1 = Integer.parseInt(param);
else
marks_1 = 0;
} catch(NumberFormatException e)
```

```
{
marks_1 = -1;
}
Result = getParameter("Result");
param = getParameter("next_class");
if(param != null)
active = Boolean.valueOf(param).booleanValue();
}
public void init()
{
}
// Display parameters.
public void paint(Graphics g)
{
g.drawString("Course Name: " + CourseName, 0, 10);
g.drawString("Marks : " + marks_1, 0, 26);
g.drawString("Result: " + Result, 0, 42);
g.drawString("Next Class: " + active, 0, 58);
}
}
```

We will get the following output:

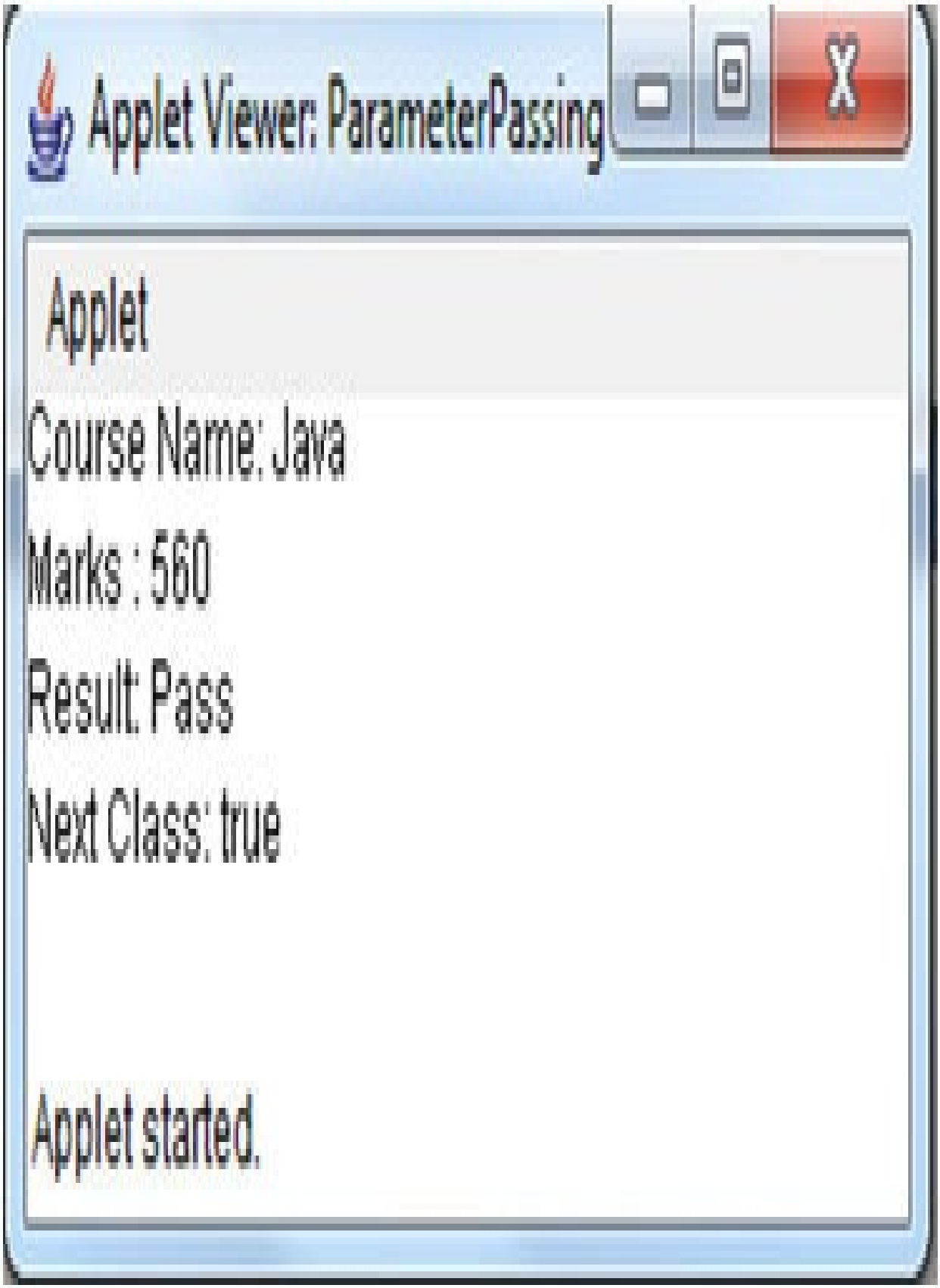

Applet Viewer: ParameterPassing
Applet
Course Name: Java
Marks : 560
Result: Pass
Next Class: true
Applet started.

Figure 8.7: A running Applet

Creating a web page with Applet

As already discussed, we can create a HTML file to include the applet tag; the same is implemented by creating a HTML file with the following code. The AppletBanner Java code is already implemented in the previous programs and as such we are only using the .class file of AppletBanner. The HTML file is as under:

```
<html>

<body bgcolor = "CYAN">

<h1> Welcome to applets through HTML</h1>

<applet code="AppletBanner" width=300 height=80>

</body >

</html>
```

The output generated after executing the above HTML file through a web browser is as under:


D:\Java\jdk1.5.0_12\bi
D:\Java\jdk1.5.0_12\bi...
Suggested Sites
Agoda.com
TripAdvisor
Welcome to applets through HTML

Figure 8.8

Handling applet events

The event handling is discussed in Chapter 10: Event Handling in Java in detail. But we can still have a look on some of the event handling programs using the applets.

The following program applet responds to the mouse events.

```
package myfirstapplet;

import java.applet.*;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.*;

import java.awt.event.*;

import java.net.URL;

import java.util.Locale;

import javax.accessibility.AccessibleContext;

public class abc extends Applet implements MouseListener

{

int x,y;
```

```
public void init()
{
Label l=new Label("Click at any place on Applet");
add(l);
addMouseListener(this);
}
public void paint(Graphics g)
{
g.drawString("*", x, y);
showStatus("You Clicked on Applet");
}
@Override
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
x= arg0.getX();
y= arg0.getY();
repaint();
}
@Override
public void mouseEntered(MouseEvent arg0) {
// TODO Auto-generated method stub
```

```
}
@Override
public void mouseExited(MouseEvent arg0) {
// TODO Auto-generated method stub


}
@Override
public void mousePressed(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseReleased(MouseEvent arg0) {
// TODO Auto-generated method stub
}
}
```

We will get the following output:

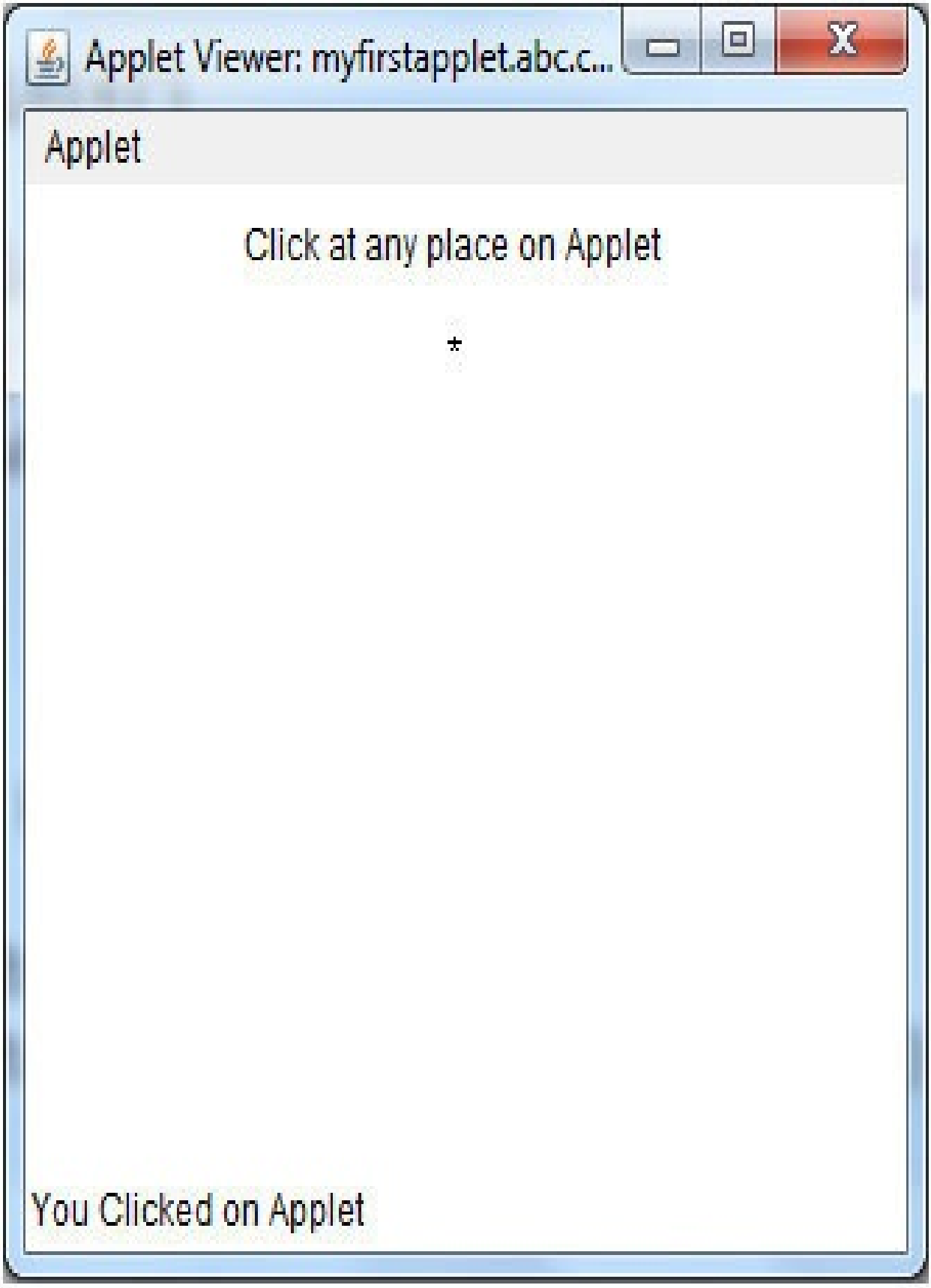

Applet Viewer: myfirstapplet.abc.c...
Applet
Click at any place on Applet
You Clicked on Applet

Figure 8.9: Running an Applet

The following applet calculates the addition and subtraction of two numbers based on the users response to two buttons:

```
package eventhandlingdemo;

import java.applet.Applet;

import java.awt.*;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

public class eventhandling extends Applet {

TextField tf;

TextField tf2;

Button bplus;

Button bMinus;

String msg;

int n1,n2,n3;
```

```
@Override
public void init() {
// TODO Auto-generated method stub
Label l1 = new Label("Enter the numbers");
tf= new TextField(20);
tf2= new TextField(20);

bplus= new Button("Addition");
bMinus= new Button("Subtraction");
add(l1);
add(tf);
add(tf2);
add(bplus);
add(bMinus);
bplus.addMouseListener( new add());
bMinus.addMouseListener(new add());
}
@Override
public void paint(Graphics g)
{
g.drawString(msg + n3, 100, 200);
```

```
}
public class add extends MouseAdapter
{
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
if (arg0.getSource() == bplus)
{
n1= Integer.parseInt(tf.getText());
n2= Integer.parseInt(tf2.getText());
n3=n1+n2;
msg ="Addition =";
}
else
{
n1= Integer.parseInt(tf.getText());
n2= Integer.parseInt(tf2.getText());
n3=n1-n2;
msg ="Subtraction =";
}
repaint();
}
```

```
}

}

}
```

We will get the following output:

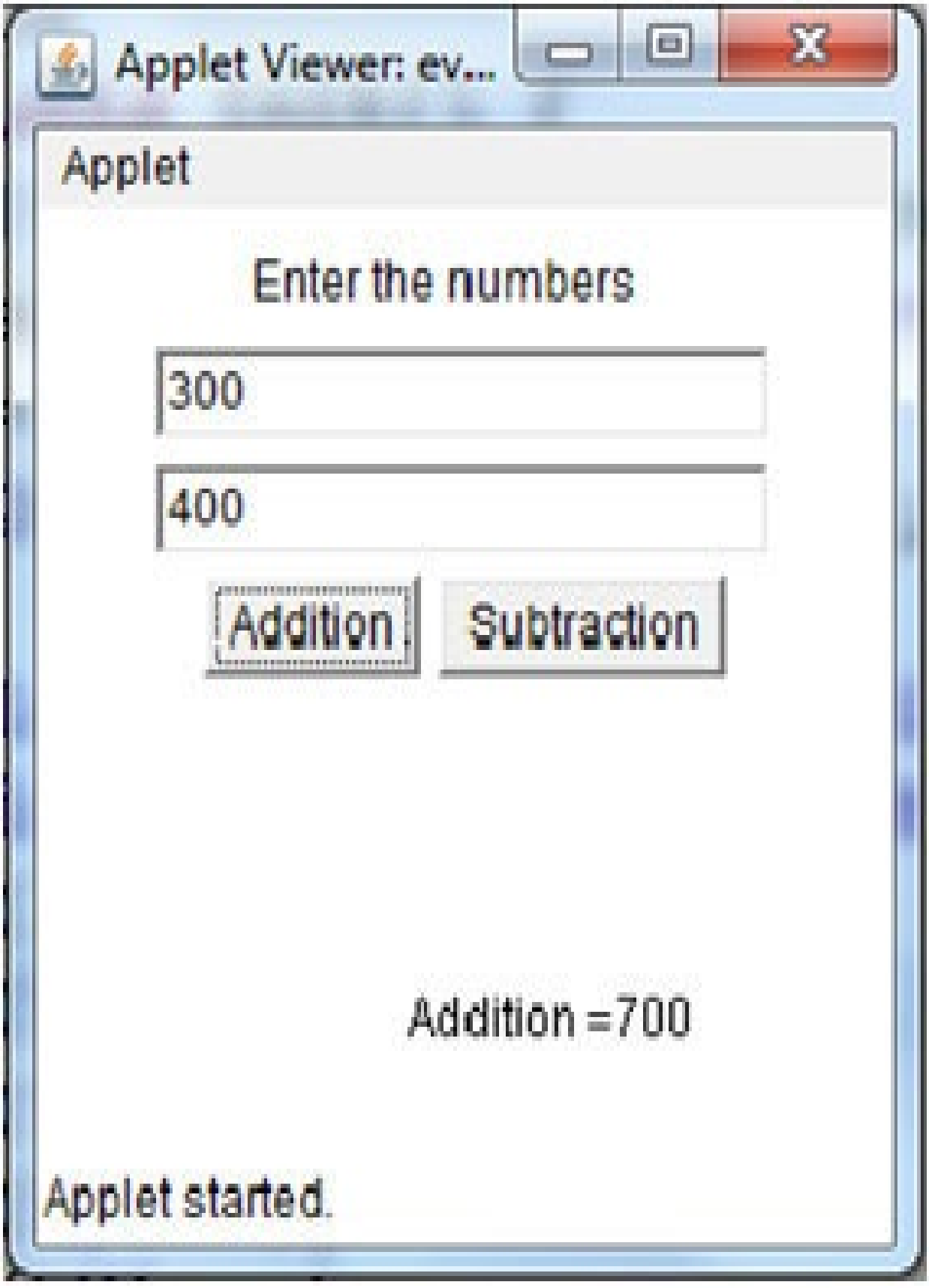

Applet Viewer: ev...
Applet
Enter the numbers
300
400
Addition
Subtraction
Addition =700
Applet started.

Figure 8.10

Assignments

How are applets different from a normal Java application programs?

Explain the life cycle of an applet? Make use of a showStatus() method to display the various states of an applet?

Create an applet that performs various string operations on a text typed in a text box?

Explain the PARAM tag? Write a program that receives two parameters from a HTML page?

Develop an applet that performs all the operations of a calculator?

CHAPTER 9

Input and Output in Java

Introduction

In this chapter we are going to discuss one of the most important packages in Java called the java.io package. As the name indicates, we are going to deal with input and output through our Java programs. So far we have not used the input from the keyboard, the reason being the lack of knowledge about packages and classes. Now we are fully aware about the inbuilt packages in Java and in this chapter we will explore the java.io package. The input from a user constitutes the main part of any Java project for example to get the details from a student about his result and all that we need is to get the input from the user. The Java makes use of GUI components to get the input from the user, but in this chapter we will explore the use of console to get the input from the user. The use of streams as in other languages like C++ is a fundamental component to get input and generate output to the user. This chapter also explores the use of files to store the data permanently and to read the data through the reading and writing operations.

Java does offer flexible support for I/O as it relates to files and networks. Java's I/O system is organized and reliable. In fact, once you comprehend its essentials, the rest of the I/O system is effortless to master.

Structure

Introduction

Streams in Java

Byte stream classes

Character stream classes

File handling in Java

Creating, reading, and writing files

Assignments

Objective

This chapter focuses on the various streams available in java to deal with the input and output. The file handling being the important concept in any of the programming language is also demonstrated in this chapter.

Streams in Java

The input/output in Java is executed through streams. A stream is an abstraction that either generates or consumes information. A stream is connected to an objective device by the Java. Every stream in Java I/O performs in the similar approach, even if the real physical devices to which they are connected, differ. Therefore, the identical I/O classes and methods can be useful to several type of devices. That indicates an input stream can abstract several diverse kinds of input: through a keyboard, a file or a socket. Similarly, an output stream may perhaps pass on to the console, a file, a printer or a network connection:

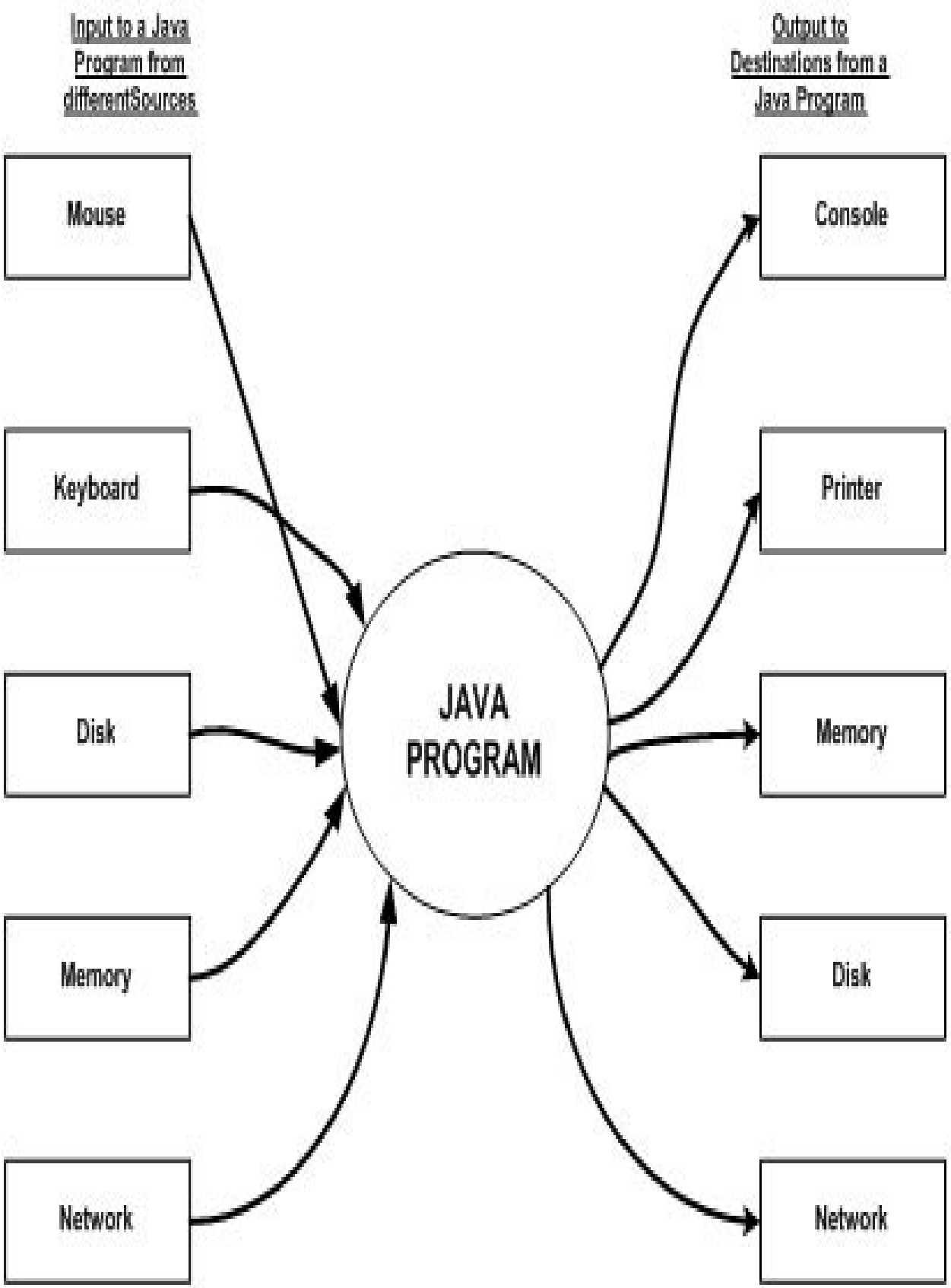

Input to a Java Program from differentSources
Output to Destinations from a Java Program
Mouse
Keyboard
Disk
Memory
Network
JAVA PROGRAM
Console
Printer
Memory
Disk
Network

Figure 9.1: Input and output devices

So a Java program reads the stream of data from the input devices through input stream classes and writes the output to the output devices through output stream classes. The java.io package contains a large number of input and output stream classes. These classes can be broadly categorized into the Byte streams and Character streams classes. Byte streams provide a convenient means for handling the input and output of bytes. Byte streams are used, for example, when reading or writing binary data. Character streams offer a suitable ways for handling input and output of characters.

The byte stream classes

Byte streams are defined by using two class hierarchies. On the pinnacle are two abstract classes: InputStream and OutputStream. Each of these abstract classes has a number of solid subclasses that handle the differences between various devices, such as files, network connections, and even memory buffers. The byte stream classes are shown in Table 9.1:

Stream class	Meaning
BufferedInputStream	Buffered input stream.
BufferedOutputStream	Buffered output stream.
ByteArrayInputStream	Input stream that reads from a byte array.
ByteArrayOutputStream	Output stream that writes to a byte array.
DataInputStream	An input stream that contains methods for reading the
DataOutputStream	An output stream that contains methods for writing the
FileInputStream	Input stream that reads from a file.
FileOutputStream	Output stream that writes to a file.
FilterInputStream	Implements InputStream.
FilterOutputStream	Implements OutputStream.
InputStream	Abstract class that describes stream input.
ObjectInputStream	Input stream for objects.
ObjectOutputStream	Output stream for objects
OutputStream	Abstract class that describes stream output
PipedInputStream	Input pipe.
PipedOutputStream	Output pipe.

PrintStream	Output stream that contains print() and println().
PushbackInputStream	Input stream that supports one-byte unget, which retur
RandomAccessFile	Supports random access file I/O.
SequenceInputStream	Input stream that is a combination of two or more inpu

Table 9.1: The byte stream classes

InputStream is an abstract class that defines Java's model of streaming byte input. Most of the methods in this class will throw an IOException on error conditions. The methods in InputStream are listed in Table 9.2:

Method	Description
int available( )	Returns the number of bytes of
void close( )	Closes the input source. Further
void mark(int numBytes)	Places a mark at the current poi
boolean markSupported( )	Returns true if mark( )/reset( ) a
int read( )	Returns an integer representatio
int read(byte buffer[ ])	Attempts to read up to buffer.le
int read(byte buffer[ ], int offset, int numBytes)	Attempts to read up to numByte
void reset( )	Resets the input pointer to the p
long skip(long numBytes)	Ignores (that is, skips) numByte

Table 9.2: The methods defined by InputStream

OutputStream is an abstract class that defines streaming byte output. Table 9.3 below shows the methods in OutputStream:

Method	Description
void close( )	Closes the output stream. Fur
void flush( )	Finalizes the output state so t
void write(int b)	Writes a single byte to an out
void write(byte buffer[ ])	Writes a complete array of by
void write(byte buffer[ ], int offset, int numBytes)	Writes a subrange of numByt

Table 9.3: The methods defined by OutputStream

The character stream classes

Character streams are defined via two class hierarchies. At the top are two abstract classes, Reader and Writer. These abstract classes handle Unicode character streams. Java has several concrete subclasses of each of these. The character stream classes are shown in the table below:

Stream class	Meaning
BufferedReader	Buffered input character stream.
BufferedWriter	Buffered output character stream.
CharArrayReader	Input stream that reads from a character array.
CharArrayWriter	Output stream that writes to a character array.
FileReader	Input stream that reads from a file.
FileWriter	Output stream that writes to a file.
FilterReader	Filtered reader.
FilterWriter	Filtered writer.
InputStreamReader	Input stream that translates bytes to characters.
LineNumberReader	Input stream that counts lines.
OutputStreamWriter	Output stream that translates characters to bytes.
PipedReader	Input pipe.
PipedWriter	Output pipe.
PrintWriter	Output stream that contains print() and println().
PushbackReader	Input stream that allows characters to be returned to the inp
Reader	Abstract class that describes character stream input.

StringReader	Input stream that reads from a string.
StringWriter	Output stream that writes to a string.
Writer	Abstract class that describes character stream output.

Table 9.4: Character stream classes defined in java.io

Reader is an abstract class that defines Java's model of streaming character input. Table 9.5 provides an outline of the methods in Reader:

Method	Description
abstract void close( )	Closes the input source.
void mark(int numChars)	Places a mark at the cur
boolean markSupported( )	Returns true if mark()/r
int read( )	Returns an integer repre
int read(char buffer[ ])	Attempts to read up to b
abstract int read(char buffer[ ],int offset, int numChars)	Attempts to read up to n
boolean ready( )	Returns true if the next
void reset( )	Resets the input pointer
long skip(long numChars)	Skips over numChar ch

Table 9.5: The methods defined by Reader

Writer is an abstract class that defines streaming character output. The various methods of Writer class are as under:

Method	Description
Writer append(char ch)	Appends ch to the en
Writer append(CharSequence chars)	Appends chars to the
Writer append(CharSequence chars, int begin, int end)	Appends the subrang
abstract void close( )	Closes the output str
abstract void flush( )	Finalizes the output s
void write(int ch)	Writes a single chara
void write(char buffer[ ])	Writes a complete ar
abstract void write(char buffer[ ], int offset, int numChars)	Writes a subrange of
void write(String str)	Writes str to the invo
void write(String str, int offset, int numChars)	Writes a subrange of

Table 9.6: The methods defined by Writer

The following interfaces are defined in java.io:

Closeable	FileFilter	ObjectInputValidation
DataInput	FilenameFilter	ObjectOutput
DataOutput	Flushable	ObjectStreamConstants
Externalizable	ObjectInput	Serializable

Table 9.7: Interfaces in java.io

Two important classes, which improve the performance by buffering input and output, are the BufferedReader and BufferedWriter. The BufferedReader has two constructors:

BufferedReader(Reader inputStream)

BufferedReader(Reader inputStream, int bufSize)

The BufferedWriter has these two constructors:

BufferedWriter(Writer outputStream)

BufferedWriter(Writer outputStream, int bufSize)

The following examples illustrates the use of the above classes

This program uses the BufferedReader class which in turn uses the InputStreamReader and System.in as its arguments to the constructor so as to get the input from the user. The read method of BufferedReader is used in this program:

```
package book;

import java.io.*;

class InputExample {
```

```
public static void main(String args[])
{
char in;
try
{
BufferedReader br = new
BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Anything you want, '0' to quit.");
do {
in = (char) br.read();
System.out.println(in);
} while(in != '0');
}
catch(IOException e)
{
System.out.println("Error While reading");
}
}
}
```

We will get the following output:

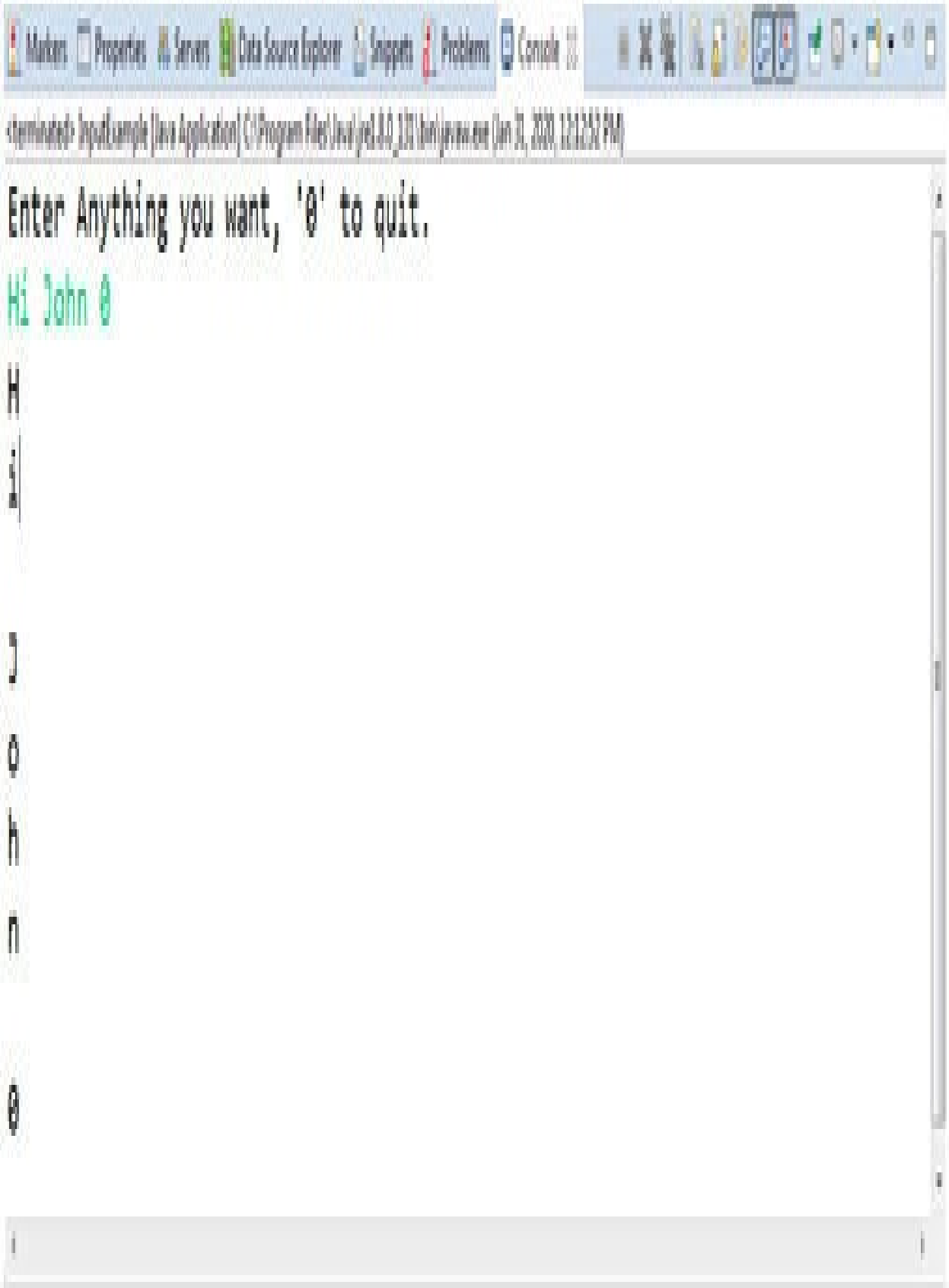

Markers
Properties
Servers
Data Source Explorer
Snippets
Problems
Console
Enter Anything you want, 'e' to quit.
Hi John e

Figure 9.2

The following program uses the BufferedReader class which in turn uses the InputStreamReader and System.in as its arguments to the constructor so as to get the input from the user in the form of a line by making use of a readLIne method.

```
package book;

import java.io.*;

class InputExamples {

public static void main(String args[]) throws IOException

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str;

System.out.println("Please Write Something");

System.out.println("Enter 'exit' to quit.");

do {

str = br.readLine();

System.out.println(str);

} while(!str.equals("stop"));

}
```

}

We will get the following output:

Please Write Something

Enter 'exit' to quit.

Hello Dear

Hello Dear

How are you

How are you

stop

stop

File handling in Java

A File class is a key class in java.io package. The object of File class is used to achieve or operate the information connected with a file on a disk. For instance, the permissions, time, date, and directory path, and to navigate the subdirectory hierarchies.

The following constructors can be used to create File objects:

File(String directoryPath)

File(String directoryPath, String filename)

File(File dirObj, String filename)

File(URI uriObj)

Some of the useful File methods are as under:

Method	Description
String getName()	Gets the name of a file.
String getPath( )	Gets the path of a file.
String getAbsolutePath ( )	Gets the absolute path of a file.
String getParent ( )	Gets the parent directory of a file.
boolean exists( )	Returns True if the file exists, otherwis

boolean isFile( )	Returns True if it is a File.
long length ( )	Returns the length of a File.
void deleteOnExit( )	Removes the file associated with the in
long getFreeSpace( )	Returns the number of free bytes of sto
long getTotalSpace( )	Returns the storage capacity of the part
long getUsableSpace( )	Returns the number of usable free bytes
boolean isHidden( )	Returns true if the invoking file is hidde
boolean setLastModified(long millisec)	Sets the time stamp on the invoking file
boolean setLastModified(long millisec)	Sets the invoking file to read-only.

Table 9.8: Methods of File class

As already discussed in the previous section, we can make use of stream classes to read and write data to different devices including files. The two important classes used in file handling are the FileInputStream and FileOutputStream.

The FileInputStream class creates an InputStream that we can use to read bytes from a file. Its two most common constructors are shown here:

FileInputStream(String filepath)

FileInputStream(File fileObj)

FileOutputStream creates an OutputStream that we can use to write bytes to a file. Its most commonly used constructors are shown here:

FileOutputStream(String filePath)

FileOutputStream(File fileObj)

FileOutputStream(String filePath, boolean append)

FileOutputStream(File fileObj, boolean append)

The following programs uses the File, FileInputStream and FileOutputStream to deal with files in Java.

Some of the important methods of File class are demonstrated in this Java program:

```
package book;

import java.io.File;

public class FileDemo {

static void p(String s) {

System.out.println(s);

}

public static void main(String args[]) {

File f1 = new File("C:/Users/NIELIT/workspace/book/demofile");

p("Name of the File " + f1.getName());

p("Path of the File " + f1.getPath());

p("Abs Path of the File: " + f1.getAbsolutePath());

p("Parent directory " + f1.getParent());

p(f1.exists() ? "Available" : "Not Available");

p(f1.canWrite() ? "is writeable" : "is not writeable");

p(f1.canRead() ? "is readable" : "is not readable");

p("is " + (f1.isDirectory() ? "" : "not" + " a directory"));

p(f1.isFile() ? "is a normal file" : "might be a named pipe");
```

```
p(f1.isAbsolute() ? "is absolute" : "is not absolute");
p("File last modified is: " + f1.lastModified());
p("File size in Bytes: " + f1.length() + " Bytes");
}
}
```

We will get the following output:

Name of the File demofile

Path of the File C:\Users\NIELIT\workspace\book\demofile

Abs Path of the File: C:\Users\NIELIT\workspace\book\demofile

Parent directory C:\Users\NIELIT\workspace\book

Available

is writeable

is readable

is not a directory

is a normal file

is absolute

File last modified is: 1572159250040

File size in Bytes: 33 Bytes

The following Java program reads from one file and writes on another file:

```
package book;
import java.io.*;
public class FileCopyExample {
public static void main(String args[]) throws IOException
{
int i;
FileInputStream fin;
FileOutputStream fout;
try {
try {
fin = new FileInputStream(args[0]);
}
catch(FileNotFoundException e)
{
System.out.println("File Not Found Exception");
return;
}
try {
fout = new FileOutputStream(args[1]);
```

```
}
catch(FileNotFoundException e) {
System.out.println("File Not Found Exception");
return;
}
} catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Please Write the File Names");
return;
}
try {
do {
i = fin.read();
if(i != -1) fout.write(i);
} while(i != -1);
} catch(IOException e)
{
System.out.println("File Handling Exception");
}
fin.close();
fout.close();
```

```
}

}
```

As we discussed the character stream classes called the Writer and Reader. The FileReader class creates a Reader that we can use to read the contents of a file. Its two mainly used constructors are shown here:

```
FileReader(String filePath)

FileReader(File fileObj)
```

Either of them can throw a FileNotFoundException.

The following example shows how to read lines from a file and print these to the standard output stream. It reads its own source file, which must be in the current directory:

```
import ava.io.*;

public class ReaderWrterDemo {

public static void main(String args[]) throws IOException

{

FileReader fr = new FileReader("Demo.java");

BufferedReader br = new BufferedReader(fr);

String s;
```

```
while((s = br.readLine()) != null)

{

System.out.println(s);

}

fr.close();

}

}
```

The FileWriter creates a Writer that we can use to write to a file. Its most commonly used constructors are shown here:

FileWriter(String filePath)

FileWriter(String filePath, boolean append)

FileWriter(File fileObj)

FileWriter(File fileObj, boolean append)

They can throw an IOException.

The following program illustrates the use of FileWriter and FileReader classes. The two file are created by writing the text on both the files and reading the contents one-by-one:

```
package book;
import java.io.*;
public class ReaderWriterDemo {
public static void main(String args[]) throws IOException
{
String source = "Hello, Its working in Files\n ";
char buffer[] = new char[source.length()];
source.getChars(0, source.length(), buffer, 0);
FileWriter f0 = new FileWriter("demofile.txt");
// Writing All Characters to demofile
for (int i=0; i < buffer.length; i ++)
{
f0.write(buffer[i]);
}
f0.close();
// Reading the demo file
System.out.println("Reading the First File ");
FileReader fin = new FileReader("demofile.txt");
for (int i=0; i < buffer.length; i ++)
{
buffer[i]=(char)fin.read();
```

```
System.out.print(buffer[i]);
}
int l = buffer.length;
fin.close();
//Writing alternate characters to file1
FileWriter f1 = new FileWriter("file1.txt");
for (int i=0; i < buffer.length; i +=2)
{
f1.write(buffer[i]);
}
f1.close();
System.out.println("Reading the Second File with alternate characters ");
FileReader f2 = new FileReader("file1.txt");
int i;
while((i=f2.read())!=-1)
{
System.out.print((char)i);
}
f2.close();
}
}
```

We will get the following output:

Reading the First File

Hello, Its working in Files

Reading the Second File with alternate characters

Hlo t okn nFls

Assignments

Distinguish between the ByteStream and CharacterStream?

Write a Java program that accepts N strings from the user and displays the names in the sorted order?

Read the contents of one file and write the even number in second file and odd numbers in third file?

Explain the various methods used in reading and writing a file in Java?

Distinguish between the InputStream and Reader classes?

CHAPTER 10

Event Handling in Java

Introduction

One of the important areas of Java development is event handling. As in case of graphical user interface programs the user has to type in into a text box and click on different type of Buttons like submit, cancel, OK, and many more. In these scenarios, the understanding of event handling for a programmer is must. In this chapter, the packages which are helpful in implementing the event handling like java.awt, java.awt.event and java.util are explored. As the applets discussed in the previous chapters are the GUI programs that we run on internet, the handling of events in applets shall be explored.

A good number of events on which our programs will take action are produced as soon as the user responds to a graphical user interface based program. The events are passed in the form of event objects specifying the details of the event that has occurred in a variety of ways, with the detailed method reliant upon the actual event that has occurred. This chapter discusses various types of events, generated by the mouse, the keyboard, and various GUI controls.

In this chapter, we will discuss the delegation event model in detail and try to find out from the scratch, how the event works and what are the different classes and interfaces involved in event handling.

Structure

Introduction

Delegation event model

Event sources

Event listeners

Event classes

Using adapter classes

Anonymous inner classes

Assignments

Objective

This chapter provides a detailed understanding of the event handling with the objective to handle all the events in a GUI based Java programs.

Delegation event model

The most advanced approach in dealing with events is the delegation event model. The detailed diagram specifying the various classes and interfaces involved in a delegation event model is as under:

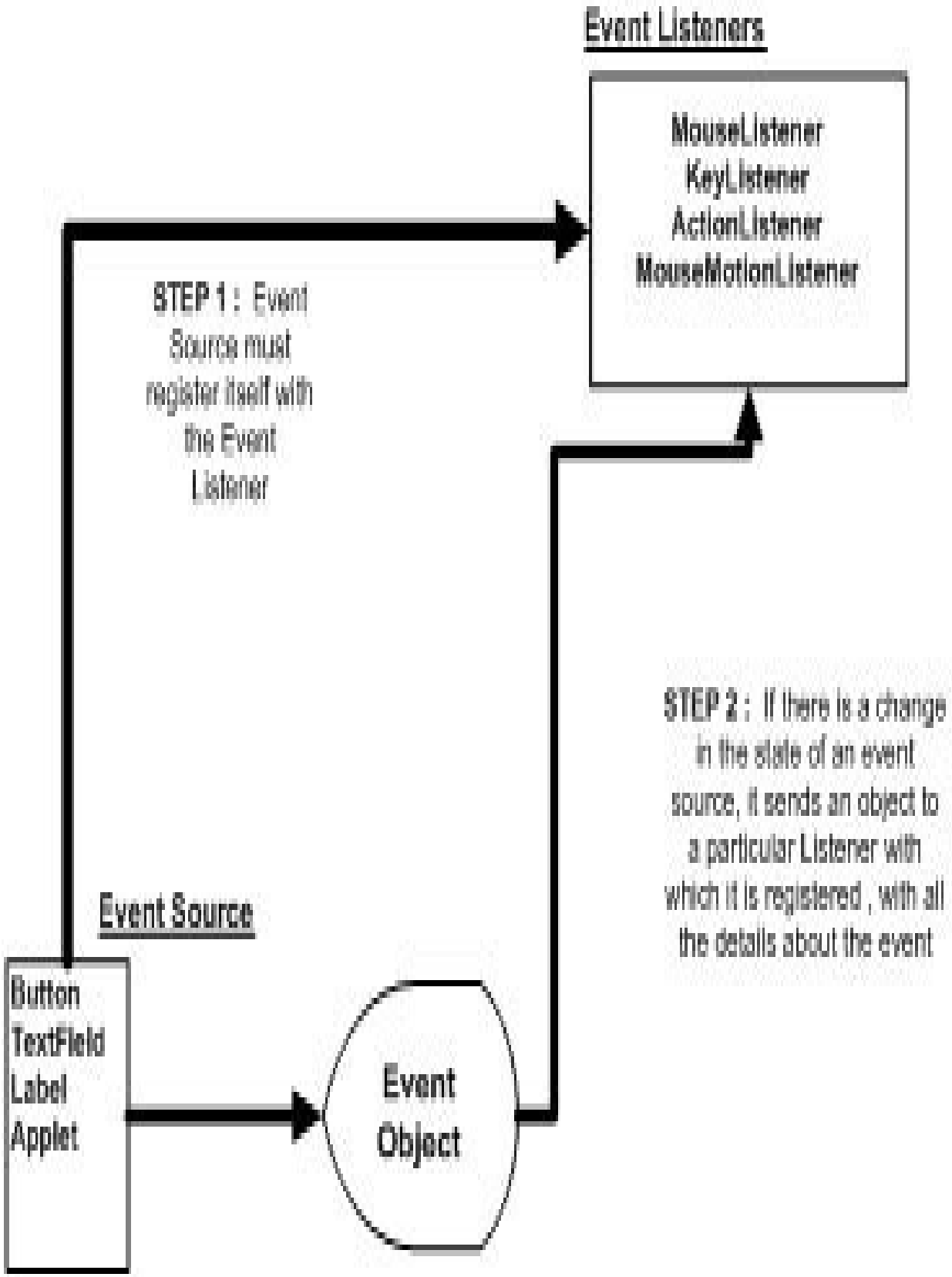

Event Listeners
MouseListener
KeyListener
ActionListener
MouseMotionListener
STEP 1 : Event Source must register itself with the Event Listener
STEP 2 : If there is a change in the state of an event source, it sends an object to a particular Listener with which it is registered , with all the details about the event
Event Source
Button
TextField
Label
Applet
Event Object

Figure 10.1: Delegation event model

The first thing that we need to understand is an event. In simple terms an event is a change in the state of an event source for example a check box is unchecked and we changed its state from unchecked to checked. These event sources must register themselves with a particular Listener. For example if we are interested in handling the Button object so that it should respond to a click event, it must register itself with a MouseListener interface. If there is any change in the state of a button it will send all the details to a Listener with which it is registered. A Listener is a class that implements an interface having abstract methods and final data. All the components of a delegation event model are explained one by one in the subsequent sections.

Event sources

In the delegation model, an event is an object that explains a state change in a source. The event gets its existence as a result of a user accessing the elements in a graphical user interface. A number of actions that are the reasons for generating an events are clicking a button, typing a character by the use of the keyboard, choosing an item in a list of item, and clicking the left, right or center button on a mouse.

Therefore, an event source is the origin of an event that is from an event generated by changing the state of the event source. An event source is an object that generates an event. This happens while the internal state of that object transforms in a little way. Event sources may perhaps generate more than one type of event.

The most important step in event handling is the registration that is the event sources must register themselves with the Listener. The general form of registration is:

public void addTypeListener(TypeListener evli)

In this general form, the type is the name of the event, and evli is a reference to the event listener. For example, the method that registers a mouse event listener is called addMouseListener(). The method that registers an action listener is called addActionListener(). When an event triggers, all registered listeners are notified and collect a copy of the event object. For example, if we want to handle the mouse events of a Button object and a TextField object, then both the objects need to register themselves with a MouseListener as under:

```
Button b1 =new Button ( "OK");

TextField tf = new TextField( );

b1.addMouseListener( this);

tf.addMouseListener (this);
```

So, the first step in delegation event model is to register the event sources with the listeners. Some of the event sources in graphical user interface are listed in the table below:

Event source	Description
Button	Generates action events when the button is pressed.
Check box	Generates item events when the check box is selected or desel
Choice	Generates item events when the choice is changed.
List	Generates action events when an item is double-clicked; gene
Menu Item	Generates action events when a menu item is selected; genera
Scroll bar	Generates adjustment events when the scroll bar is manipulate
Text components	Generates text events when the user enters a character.
Window	Generates window events when a window is activated, closed,

Table 10.1: Event sources

Event listeners

A listener is an object that is alerted as soon as an event takes place. The two most important necessities of an event listener are it must have been registered with one or more sources to receive notifications about precise types of events. Second, it should implement methods to accept and process these notifications.

For example, if a Button object registers with the MouseListener interface, it must implement all of its methods that accept and process events. The methods are defined in a set of interfaces found in java.awt.event. For example, the MouseListener interface defines five methods to accept notifications when the mouse is clicked:

```
void mouseClicked(MouseEvent me)

{

// Action to be taken when the mouse gets clicked

}

void mouseEntered(MouseEvent me)

{

// Action to be taken when the mouse enter within the control

}

void mouseExited(MouseEvent me)

{
```

```
// Action to be taken when the mouse exits the control

}

void mousePressed(MouseEvent me)

{

// Action to be taken when the mouse button is pressed

}

void mouseReleased(MouseEvent me)

{

// Action to be taken when the mouse button is released

}
```

The below table lists the interfaces available along with the methods that need to be implemented by the event listener:

Interface	Methods
ActionListener	
AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent ae)
ComponentListener	void componentResized(ComponentEvent ce) void com
ContainerListener	void componentAdded(ContainerEvent ce) void compor
FocusListener	void focusGained(FocusEvent fe) void focusLost(Focus
ItemListener	void itemStateChanged(ItemEvent ie)
KeyListener	void keyPressed(KeyEvent ke) void keyReleased(KeyEv
MouseListener	void mouseClicked(MouseEvent me) void mouseEntere

MouseMotionListener	void mouseDragged(MouseEvent me) void mouseMove
MouseWheelListener	void mouseWheelMoved(MouseWheelEvent mwe)
TextListener	void textChanged(TextEvent te)
WindowFocusListener	void windowGainedFocus(WindowEvent we) void wind
WindowListener	void windowActivated(WindowEvent we) void window

Table 10.2: Event listeners with abstract methods

Event classes

Once the object is registered with a listener, it sends a notification to the methods within that listener. While notifying the method, an event object is sent to the listener, which encapsulates the details of the event that has taken place. So the event objects are at the core of event handling in Java. For example, if we have registered our TextField object with a KeyListener interface as under:

```
TextField tf = new TextField( );

tf.addKeyListener (this);
```

The KeyListener interface has three methods, which are to be implemented by the event listener. All the three methods must be implemented as under:

```
void keyPressed(KeyEvent ke)

{

}

void keyReleased(KeyEvent ke)

{

}

void keyTyped(KeyEvent ke)

{
```

}

Once we press any key in the TextField object, an event gets triggered and an object of a KeyEvent class is sent to all the three methods. It provides us the exact details of the event that has occurred, say for example, we are interested in knowing the keyboard key that was typed by the user, we can use the object of KeyEvent class as:

```
char ch = ke.getKeyChar( );
```

The various Event classes defined in java.util are listed in table below:

Event class	Description
ActionEvent	Generated when a button is pressed, a list item is double-cli
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or
ContainerEvent	Generated when a component is added to or removed from a
FocusEvent	Generated when a component gains or loses keyboard focus
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occ
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pres
MouseWheelEvent	Generated when the mouse wheel is moved.
TextEvent	Generated when the value of a text area or text field is chan
WindowEvent	Generated when a window is activated, closed, deactivated,

Table 10.3: Event classes

At the core of the Java event class ladder is EventObject available in java.util. It is the superclass for all events. It's one constructor is shown here:

EventObject(Object src)

EventObject contains two methods: getSource() and toString(). The getSource() method returns the source of the event. Its general form is shown here:

Object getSource( )

The toString() returns the string equivalent of the event.

AWTEvent is a superclass of all AWT events that are handled by the delegation event model. The AWTEvent is defined within the java.awt package and is a subclass of EventObject.

One of the important method of AWTEvent is getID() which gets the type of the event. The general form of this object is:

int getID( )

The detailed descriptions of event classes defined in java.awt.event are as under:

Event class	Constructors
ActionEvent	ActionEvent(Object src, int type, String cmd) ActionEvent(
AdjustmentEvent	AdjustmentEvent(Adjustable src, int id, int type, int data)
ComponentEvent	ComponentEvent(Component src, int type)
ContainerEvent	ContainerEvent(Component src, int type, Component comp
FocusEvent	FocusEvent(Component src, int type) FocusEvent(Compone
InputEvent	abstract class inherited by KeyEvent and MouseEvent
ItemEvent	ItemEvent(ItemSelectable src, int type, Object entry, int stat
KeyEvent	KeyEvent(Component src, int type, long when, int modifier
MouseEvent	MouseEvent(Component src, int type, long when, int modif
MouseWheelEvent	MouseWheelEvent(Component src, int type, long when, in
TextEvent	TextEvent(Object src, int type)
WindowEvent	WindowEvent(Window src, int type) WindowEvent(Windo

Table 10.4: Event classes with constructors, methods and constants.

In the following Java program a GUI is created and the controls are responding to various keyboard and mouse events as depicted by the output:

```
package book;

import java.awt.*;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import javax.swing.JOptionPane;

public class GUIFormWithEvents implements MouseListener, KeyListener

{

static TextField txtname;

static TextField txtenrolment;

static Button btnsubmit;

static Button btncancel;

String msg;

String name;
```

```
int en_no;
static Frame f = null;
public static void main(String arg[]) {
f = new Frame("Calculator….");
Label l1 = new Label("Please Fill the Form Correctly");
Label l2 = new Label("Enter Name");
Label l3 = new Label("Enter Enrolment");
// Label Objects are used within the Frame
l1.setBackground(Color.BLUE);
txtname= new TextField(20);
txtenrolment= new TextField(20);
btnsubmit= new Button("Submit");
btncancel= new Button("Cancel");
f.setLayout(null);
f.setBackground(Color.CYAN);
f.add(l1).setBounds(150, 50, 340, 40);
f.add(l2).setBounds(150, 100, 100, 40);
f.add(l3).setBounds(150, 150, 100, 40);
f.add(txtname).setBounds(300, 100, 175, 30);
f.add(txtenrolment).setBounds(300, 150, 175, 30);
f.add(btnsubmit).setBounds(350, 250, 100, 30);
```

```
f.add(btncancel).setBounds(200, 250, 100, 30);
txtname.addKeyListener(new GUIFormWithEvents());
txtenrolment.addKeyListener(new GUIFormWithEvents());
btnsubmit.addMouseListener( new GUIFormWithEvents());
btncancel.addMouseListener(new GUIFormWithEvents());
f.setSize(500, 500);
f.setVisible(true);
}
@Override
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
name= txtname.getText();
en_no= Integer.parseInt(txtenrolment.getText());
if (arg0.getSource() == btnsubmit)
{
JOptionPane.showOptionDialog(
f,"Hello "+ name + " Your enrolment no "+ en_no +" has been submitted",
"Form Submission",
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
null,
```

```
null,
null);
}
else if (arg0.getSource() == btncancel)
{
f.setVisible(false);


}


}
@Override
public void mouseEntered(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseExited(MouseEvent arg0) {
// TODO Auto-generated method stub
}

@Override
public void mousePressed(MouseEvent arg0) {
```

```
// TODO Auto-generated method stub
}
@Override
public void mouseReleased(MouseEvent arg0) {
// TODO Auto-generated method stub

}
@Override
public void keyPressed(KeyEvent arg0) {
// TODO Auto-generated method stub

}
@Override
public void keyReleased(KeyEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void keyTyped(KeyEvent arg0)
{
if ((arg0.getSource() == txtname) && (Character.isDigit(arg0.getKeyChar())))
{
```

```
JOptionPane.showOptionDialog(

f,"Numbers Not Allowed",

"Enter a valid Name",

JOptionPane.DEFAULT_OPTION,

JOptionPane.INFORMATION_MESSAGE,

null,

null,

null);

arg0.setKeyChar((char) KeyEvent.VK_CANCEL);

}

if ((arg0.getSource() == txtenrolment) &&
(Character.isAlphabetic(arg0.getKeyChar())))

{

JOptionPane.showOptionDialog(

f,"Characters Not Allowed",

"Enter a valid Enrolment",

JOptionPane.DEFAULT_OPTION,

JOptionPane.INFORMATION_MESSAGE,

null,

null,

null);
```

```
arg0.setKeyChar((char) KeyEvent.VK_CANCEL);
}


}
}
```

We will get the following output:


Calculator...
Enter Name
Mehvish Nazir
Enter Enrolment
564
Cancel
Submit
Form Submission
Hello Mehvish Nazir Your enrolment no 564 has been submitted
OK

Figure 10.2

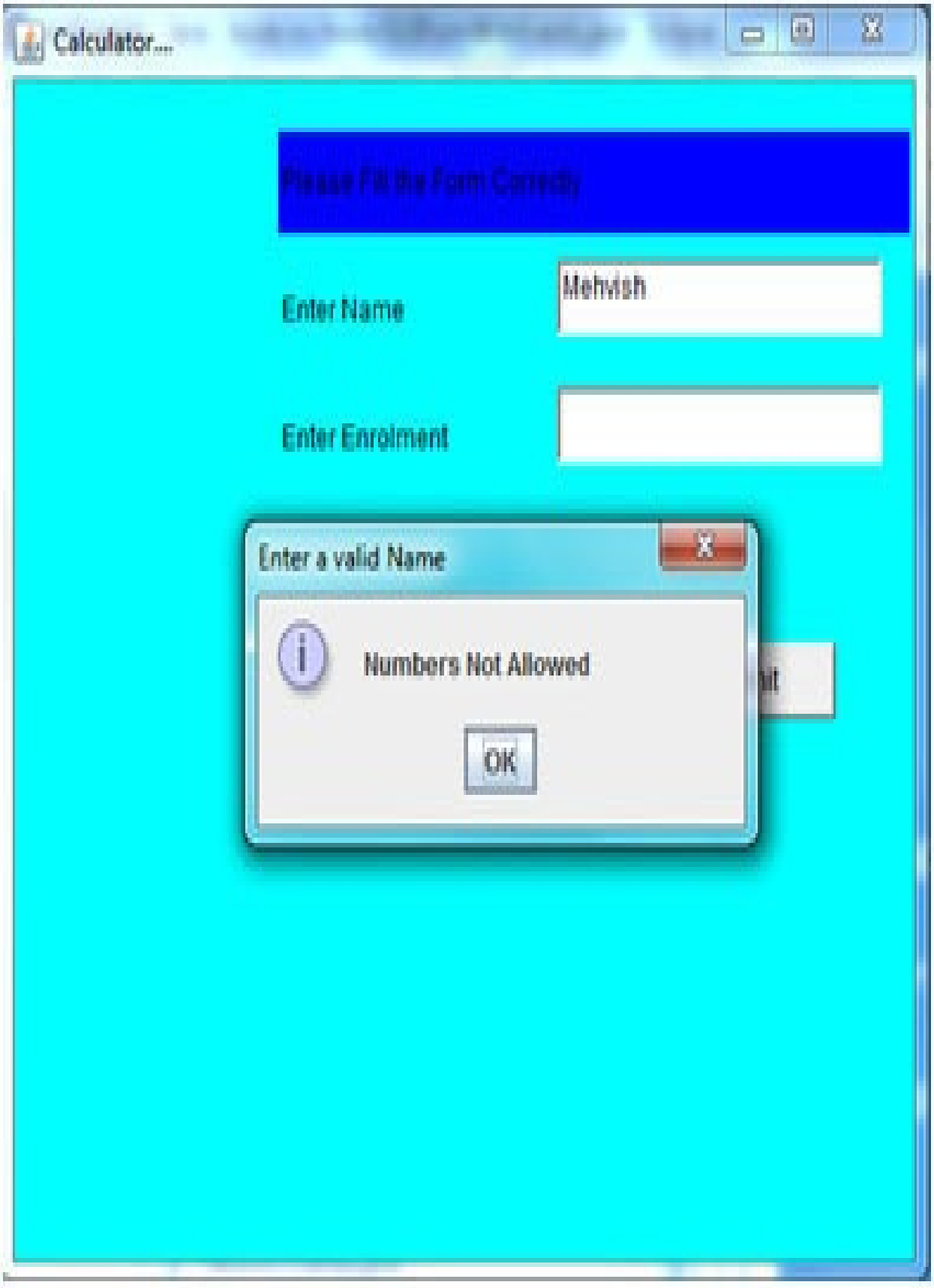

Calculator....
Please Fill the Form Correctly
Enter Name
Mehvish
Enter Enrolment
Enter a valid Name
Numbers Not Allowed
OK

Figure 10.3

The above output does not allow the numbers while entering the name and the below screenshot of the output does not allow characters while entering the enrolment number:

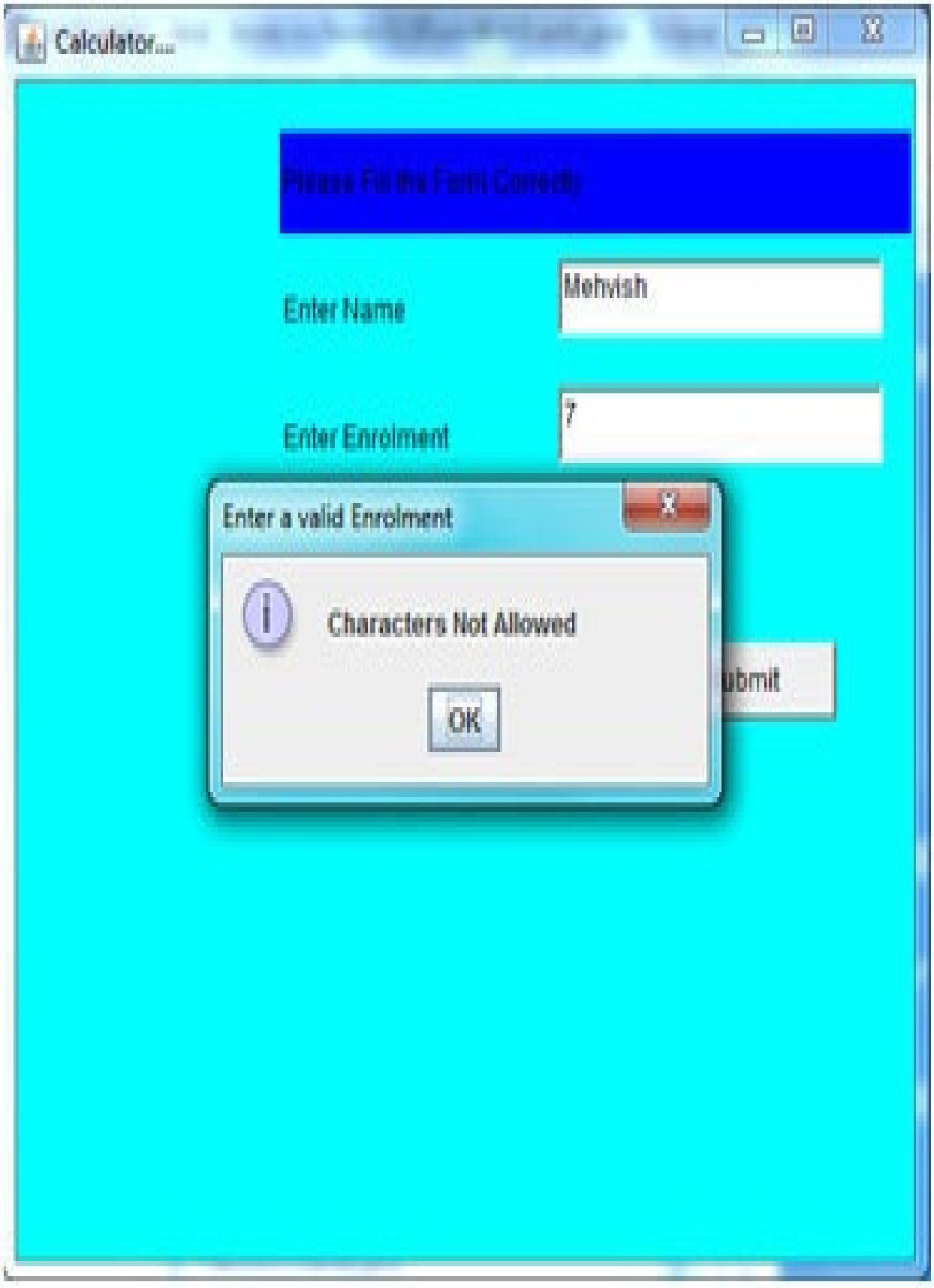

Calculator...
Please Fill the Form Correctly
Enter Name
Mehvish
Enter Enrolment
7
Enter a valid Enrolment
Characters Not Allowed
OK
ubmit

Figure 10.4

In this Java program a simple GUI is created by making use of a Frame class. The Label, TextField, and Button objects are created. The TextField object is registered with The KeyListener interface and Button object is registered with the MouseListener. The TextField responds to the keyboard, if the user presses any of the digits it rejects it by sending a proper message. Once the user types his/her name, the Button can be clicked and a welcome message is displayed:

```
package book;

import java.awt.*;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import javax.swing.JOptionPane;

public class TextListenerDemo implements MouseListener, KeyListener

{

static TextField tf=null;

static Frame jf=null;

public static void main(String[] args) {

jf = new Frame("Event Handling");
```

```
Label lbl= new Label("Enter Your Name:");
tf = new TextField();
//Register TextField With the KeyListener
tf.addKeyListener(new TextListenerDemo());
Button b1= new Button("Click Here…");
// Register Button With the MouseListener
b1.addMouseListener(new TextListenerDemo());
jf.setSize(500, 500);
jf.setLayout(null);
jf.add(lbl).setBounds(50, 50, 100, 70);;
jf.add(tf).setBounds(50, 120, 400, 30);
jf.add(b1).setBounds(200, 200, 100, 70);
jf.setVisible(true);

}
@Override
public void mouseClicked(MouseEvent arg0) {

int n = JOptionPane.showOptionDialog(
jf, "Welcome " + tf.getText(),
"A Welcome Message",
```

```
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
null,
null,
null);
}
@Override
public void mouseEntered(MouseEvent arg0) {
}
@Override
public void mouseExited(MouseEvent arg0) {
}
@Override
public void mousePressed(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mouseReleased(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
```

```
public void keyPressed(KeyEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void keyReleased(KeyEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void keyTyped(KeyEvent arg0) {

char ch =arg0.getKeyChar();
if (Character.isDigit(ch))
{
int n = JOptionPane.showOptionDialog(
jf, "Numbers are Not allowed",
"Message",
JOptionPane.PLAIN_MESSAGE,
JOptionPane.ERROR_MESSAGE,
null,
null,
null);
```

```
arg0.setKeyChar((char) KeyEvent.VK_CANCEL);
}
}
}
```

We will get the following outputs:

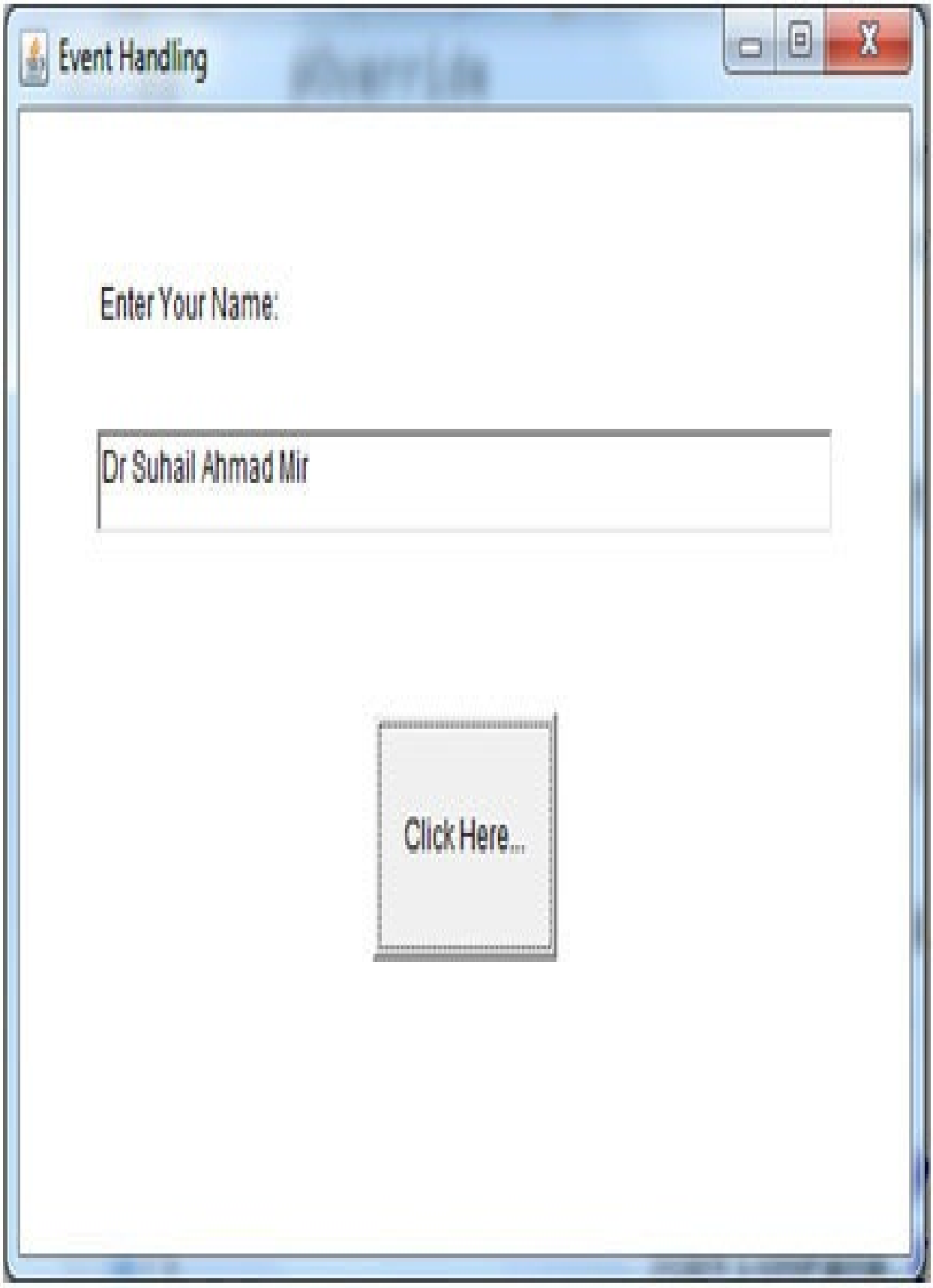

Event Handling
Enter Your Name:
Dr Suhail Ahmad Mir
Click Here...

Figure 10.5

After clicking on the above button, the welcome message pops up as in the output below:


Event Handling
Enter Your Name:
Dr Suhail Ahmad Mir
A Welcome Message
Welcome Dr Suhail Ahmad Mir
OK

Figure 10.6

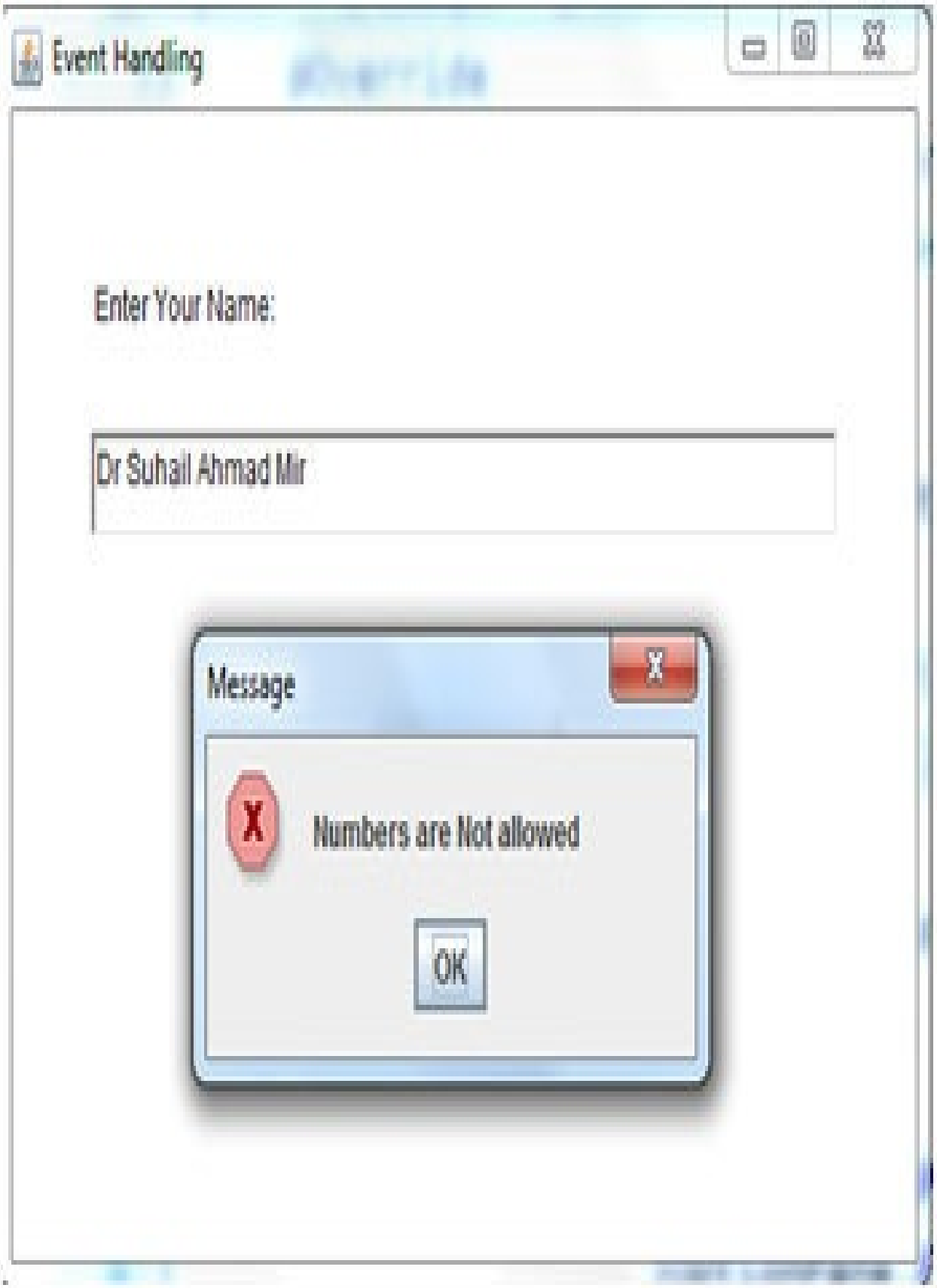

Event Handling
Enter Your Name:
Dr Suhail Ahmad Mir
Message
Numbers are Not allowed
OK

Figure 10.7

In the Java program, an applet is created and is registered with MouseListener and MouseMotionListener. All the methods of these two listeners are implemented and the paint ( ) method of applet is used to display the messages when the event triggers:

```
package book;

import java.awt.event.*;

import java.applet.*;

import java.awt.*;


/*


<applet code="MouseEvents" width=300 height=100></applet>


*/

public class EventHandlingDemo extends Applet implements MouseListener,
MouseMotionListener {

String msg = "";

int mouseX = 0, mouseY = 0; // coordinates of mouse

public void init() {
```

```
addMouseListener(this);
addMouseMotionListener(this);
setBackground(Color.RED);
}
// Handle mouse clicked.
public void mouseClicked(MouseEvent me) {
mouseX = 100; mouseY = 100;
msg = "Mouse clicked."; repaint();
}
// Handle mouse entered.
public void mouseEntered(MouseEvent me) {
mouseX = 0; mouseY = 10;
msg = "Mouse entered."; repaint();
}
// Handle mouse exited.
public void mouseExited(MouseEvent me) {
mouseX = 0; mouseY = 10;
msg = "Mouse exited."; repaint();
}
// Handle button pressed.
public void mousePressed(MouseEvent me) {
```

```
mouseX = me.getX(); mouseY = me.getY();

msg = "Mouse Down Here"; repaint();

}

public void mouseReleased(MouseEvent me) {

mouseX = me.getX(); mouseY = me.getY();

msg = "Mouse Up Here"; repaint();

}

// Handle mouse dragged.

public void mouseDragged(MouseEvent me) {

mouseX = me.getX(); mouseY = me.getY();

msg = "*";

showStatus("The X & Y Position of Mouse= " + mouseX + ", " + mouseY);
repaint();

}

// Handle mouse moved.

public void mouseMoved(MouseEvent me) {

// show status

showStatus("The X & Y Position of Mouse= " + me.getX() + ", " + me.getY());

}

public void paint(Graphics g) {

g.drawString(msg, mouseX, mouseY);
```

```
}
}
```

We will get the following output:

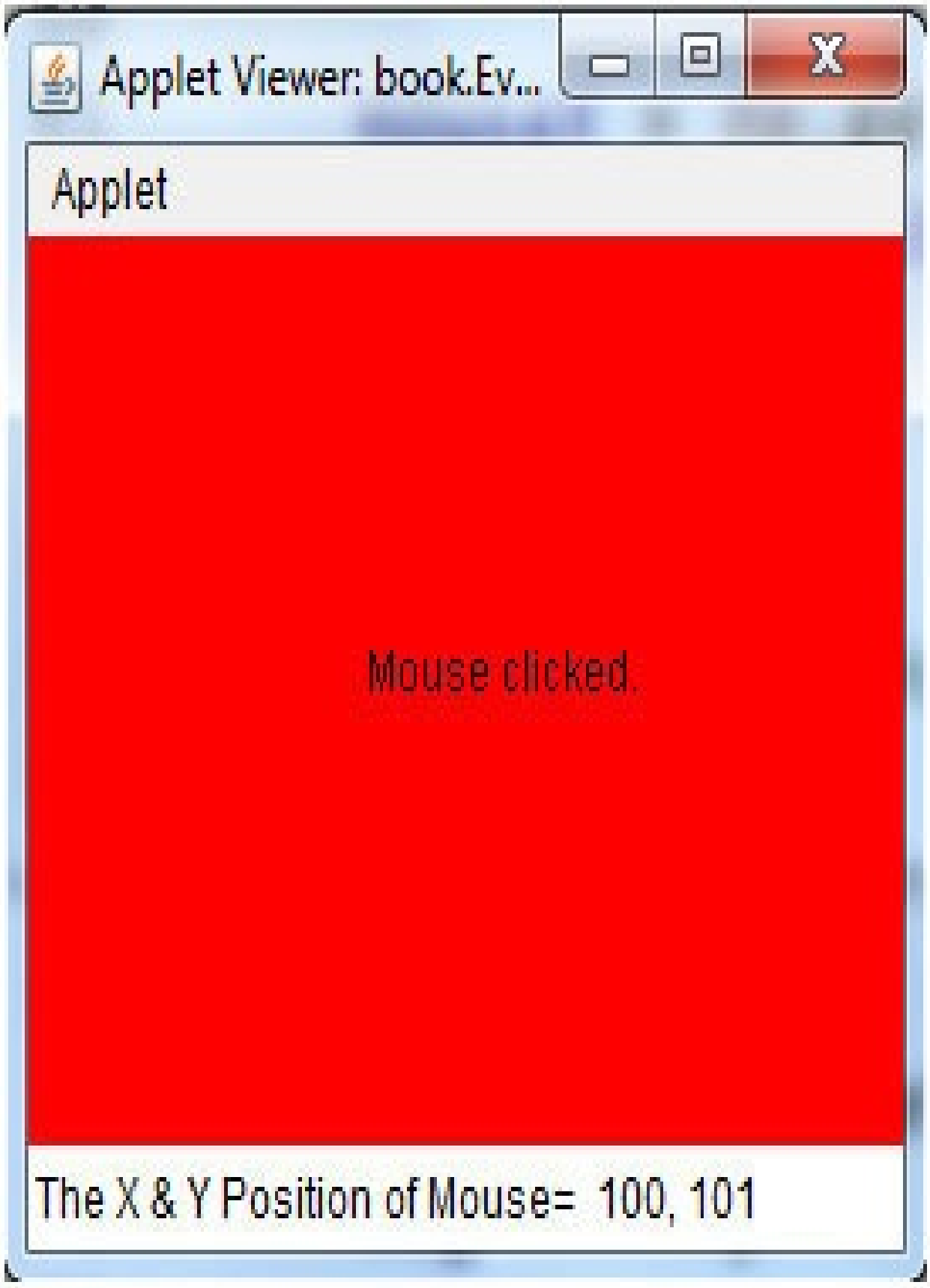

Applet Viewer: book.Ev...
Applet
Mouse clicked.
The X & Y Position of Mouse= 100, 101

Figure 10.8

In the following program, four buttons to carry out addition, subtraction, multiplication, and division are created. All the buttons are registered with MouseListener and the desired operation is performed:

```
package book;

import java.awt.*;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import javax.swing.JOptionPane;

public class CalculatorDemo implements MouseListener

{

static TextField tf;

static TextField tf2;

static Button bplus;

static Button bMinus;

static Button bdivide;

static Button bmult;

String msg;

float n1,n2,n3;
```

```
static Frame f = null;
public static void main(String arg[]) {
f = new Frame("Calculator….");
Label l1 = new Label("Enter the Numbers");
tf= new TextField(20);
tf2= new TextField(20);
bplus= new Button("Addition");
bMinus= new Button("Subtraction");
bdivide= new Button("Division");
bmult= new Button("Multiplication");
f.setLayout(null);
f.add(l1).setBounds(150, 30, 120, 70);
f.add(tf).setBounds(150, 100, 100, 30);
f.add(tf2).setBounds(250, 100, 100, 30);
f.add(bplus).setBounds(150, 150, 100, 30);
f.add(bMinus).setBounds(150, 200, 100, 30);
f.add(bdivide).setBounds(250, 150, 100, 30);
f.add(bmult).setBounds(250, 200, 100, 30);
bplus.addMouseListener( new CalculatorDemo());
bMinus.addMouseListener(new CalculatorDemo());
bdivide.addMouseListener( new CalculatorDemo());
```

```
bmult.addMouseListener( new CalculatorDemo());
f.setSize(500, 500);
f.setVisible(true);
}
@Override
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
n1= Integer.parseInt(tf.getText());
n2= Integer.parseInt(tf2.getText());
if (arg0.getSource() == bplus)
{
n3=n1+n2;
msg ="Addition =";
}
else if (arg0.getSource() == bMinus)
{
n3=n1-n2;
msg ="Subtraction =";
}
else if (arg0.getSource() == bmult)
{
```

```
n3=n1*n2;
msg ="Multiplication =";
}
else
{
n3=n1/n2;
msg ="Division =";
}
JOptionPane.showOptionDialog(
f, msg + n3,
msg,
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
null,
null,
null);
}
@Override
public void mouseEntered(MouseEvent arg0) {
// TODO Auto-generated method stub
}
```

```
@Override
public void mouseExited(MouseEvent arg0) {
// TODO Auto-generated method stub
}
@Override
public void mousePressed(MouseEvent arg0) {
// TODO Auto-generated method stub
}

@Override
public void mouseReleased(MouseEvent arg0) {
// TODO Auto-generated method stub
}
}
```

We will get the following output:

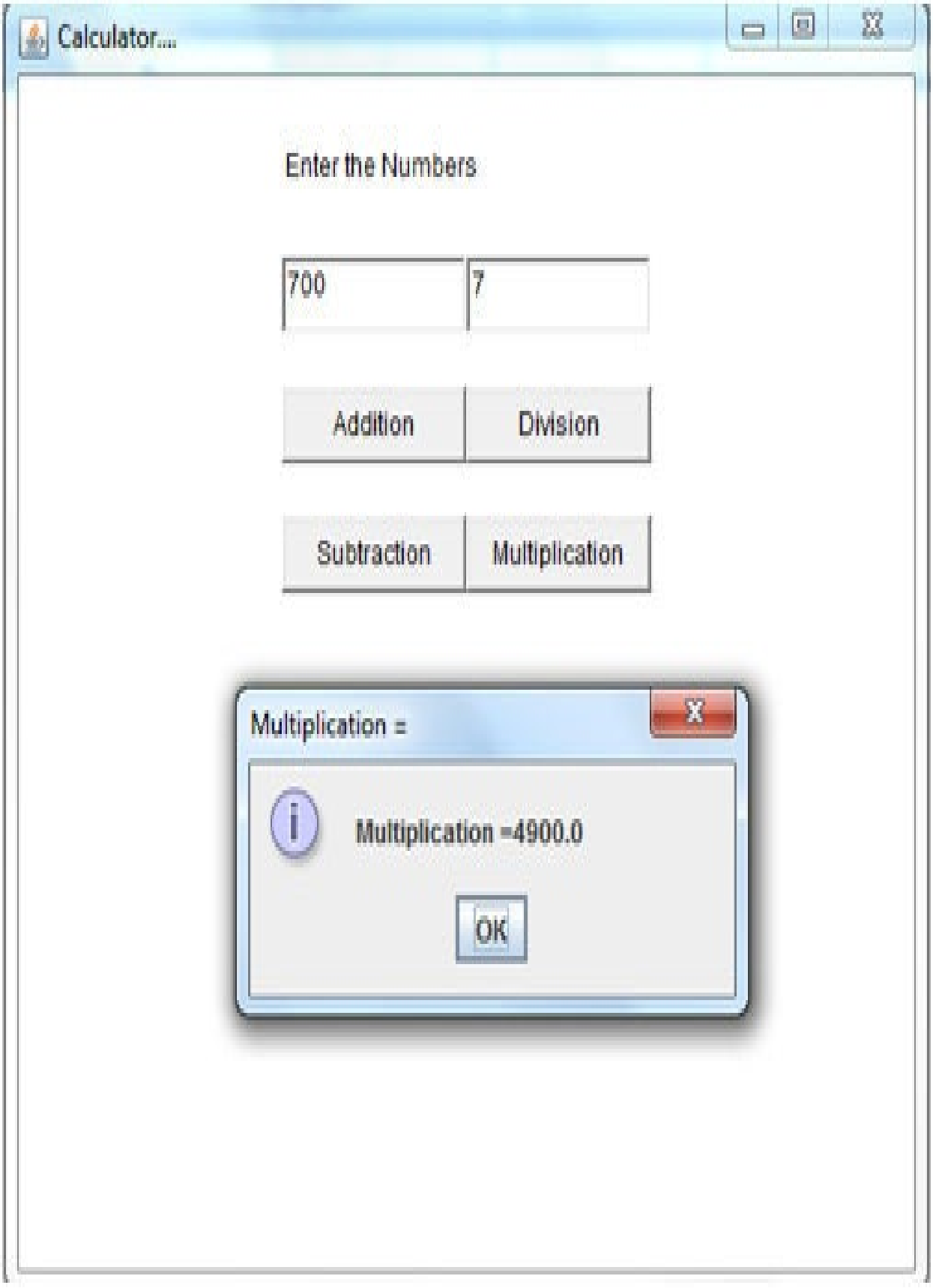

Calculator....
Enter the Numbers
700
7
Addition
Division
Subtraction
Multiplication
Multiplication =
Multiplication =4900.0
OK

Figure 10.9

Adapter classes

A unique characteristic of java, called an Adapter class that can make things easier for the construction of event handlers in certain circumstances. An Adapter class offers an unfilled execution of all methods in an event listener interface. Adapter classes are helpful when we would like to get and process simply a few of the events that are handled by a particular event listener interface. We can name a fresh class to proceed as an event listener by extending one of the adapter classes and implementing barely those events in which we are concerned. For example if we want to implement the MouseListener interface, we have to implement all of its methods which are void mouseClicked(MouseEvent me), void mouseEntered(MouseEvent me), void mouseExited(MouseEvent me), void mousePressed(MouseEvent me), void mouseReleased(MouseEvent me). But out of all these five methods, we only want to implement the mouseClicked () method to deal with the click event. We can create another class called the MyMouseAdapter that extends MouseAdapter and overrides the mouseClicked() method. The other mouse events are silently ignored by code inherited from the MouseAdapter class. The below table lists the Adapter classes used for the implementation of Listener interfaces:

Adapter class	Listener interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

Table 10.5: Adapter classes

In the following Java program the use of Adapter class is demonstrated. The MouseAdapter class is used and extended by the AdapterClass. Out of five methods of MouseListener only the mouseClicked () method is implemented:

```
package book;

import java.awt.*;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import javax.swing.JOptionPane;

public class AdapterDemo {

static TextField tf;

static TextField tf2;

static Button bplus;

static Button bMinus;

static Button bdivide;

static Button bmult;

static String msg;

static float n1;
```

```
static float n2;
static float n3;
static Frame f = null;
public static void main(String arg[]) {
f = new Frame("Calculator….");
Label l1 = new Label("Enter the Numbers");
tf= new TextField(20);
tf2= new TextField(20);
bplus= new Button("Addition");
bMinus= new Button("Subtraction");
bdivide= new Button("Division");
bmult= new Button("Multiplication");
f.setLayout(null);
f.add(l1).setBounds(150, 30, 120, 70);
f.add(tf).setBounds(150, 100, 100, 30);
f.add(tf2).setBounds(250, 100, 100, 30);
f.add(bplus).setBounds(150, 150, 100, 30);
f.add(bMinus).setBounds(150, 200, 100, 30);
f.add(bdivide).setBounds(250, 150, 100, 30);
f.add(bmult).setBounds(250, 200, 100, 30);
bplus.addMouseListener( new AdapterClass());
```

```
bMinus.addMouseListener(new AdapterClass());
bdivide.addMouseListener( new AdapterClass());
bmult.addMouseListener( new AdapterClass());
f.setSize(500, 500);
f.setVisible(true);
}
}
class AdapterClass extends MouseAdapter
{
@Override
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
AdapterDemo.n1= Integer.parseInt(AdapterDemo.tf.getText());
AdapterDemo.n2= Integer.parseInt(AdapterDemo.tf2.getText());
if (arg0.getSource() == AdapterDemo.bplus)
{
AdapterDemo.n3=AdapterDemo.n1+AdapterDemo.n2;
AdapterDemo.msg ="Addition =";
}
else if (arg0.getSource() == AdapterDemo.bMinus)
{
```

```
AdapterDemo.n3=AdapterDemo.n1-AdapterDemo.n2;
AdapterDemo.msg ="Subtraction =";
}
else if (arg0.getSource() == AdapterDemo.bmult)
{
AdapterDemo.n3=AdapterDemo.n1*AdapterDemo.n2;
AdapterDemo.msg ="Multiplication =";
}
else
{
AdapterDemo.n3=AdapterDemo.n1/AdapterDemo.n2;
AdapterDemo.msg ="Division =";
}
JOptionPane.showOptionDialog(
AdapterDemo.f, AdapterDemo.msg + AdapterDemo.n3,
AdapterDemo.msg,
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
null,
null,
null);
```

```
}
}
```

We will get the following output:

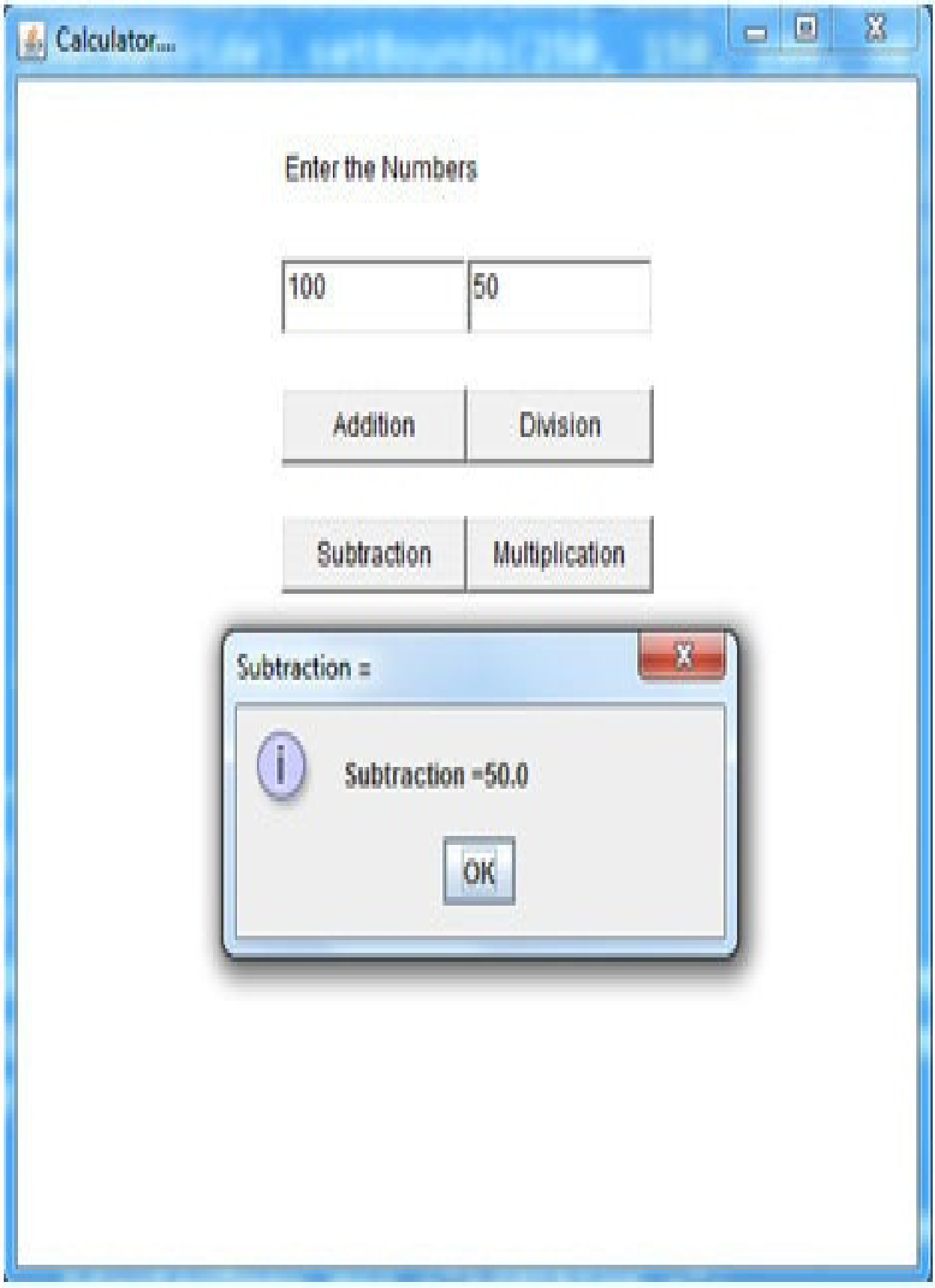

Calculator....
Enter the Numbers
100
50
Addition
Division
Subtraction
Multiplication
Subtraction =
Subtraction =50.0
OK

Figure 10.10

In the following Java program, the Adapter class is used to handle the mouse events:

```
package book;

import java.awt.*;

import java.awt.event.*;

import java.applet.*;

/*

<applet code="AdapterDemoApplets" width=300 height=100>

</applet>

*/

public class AdapterDemoApplets extends Applet

{

public void init()

{

addMouseListener(new MyMouseAdapter(this));

addMouseMotionListener(new MyMouseMotionAdapter(this));

}
```

```
}

class MyMouseAdapter extends MouseAdapter {
AdapterDemoApplets adapterDemo;
public MyMouseAdapter(AdapterDemoApplets adapterDemo)
{
this.adapterDemo = adapterDemo;
}

// Handle mouse clicked.
public void mouseClicked(MouseEvent me)
{
adapterDemo.showStatus("Mouse clicked");
}
}
class MyMouseMotionAdapter extends MouseMotionAdapter
{
AdapterDemoApplets adapterDemo;
public MyMouseMotionAdapter(AdapterDemoApplets adapterDemo)
{
this.adapterDemo = adapterDemo;
```

```
}

// Handle mouse dragged.
public void mouseDragged(MouseEvent me)
{
adapterDemo.showStatus("Mouse dragged");
}
}
```

We will get the following output:

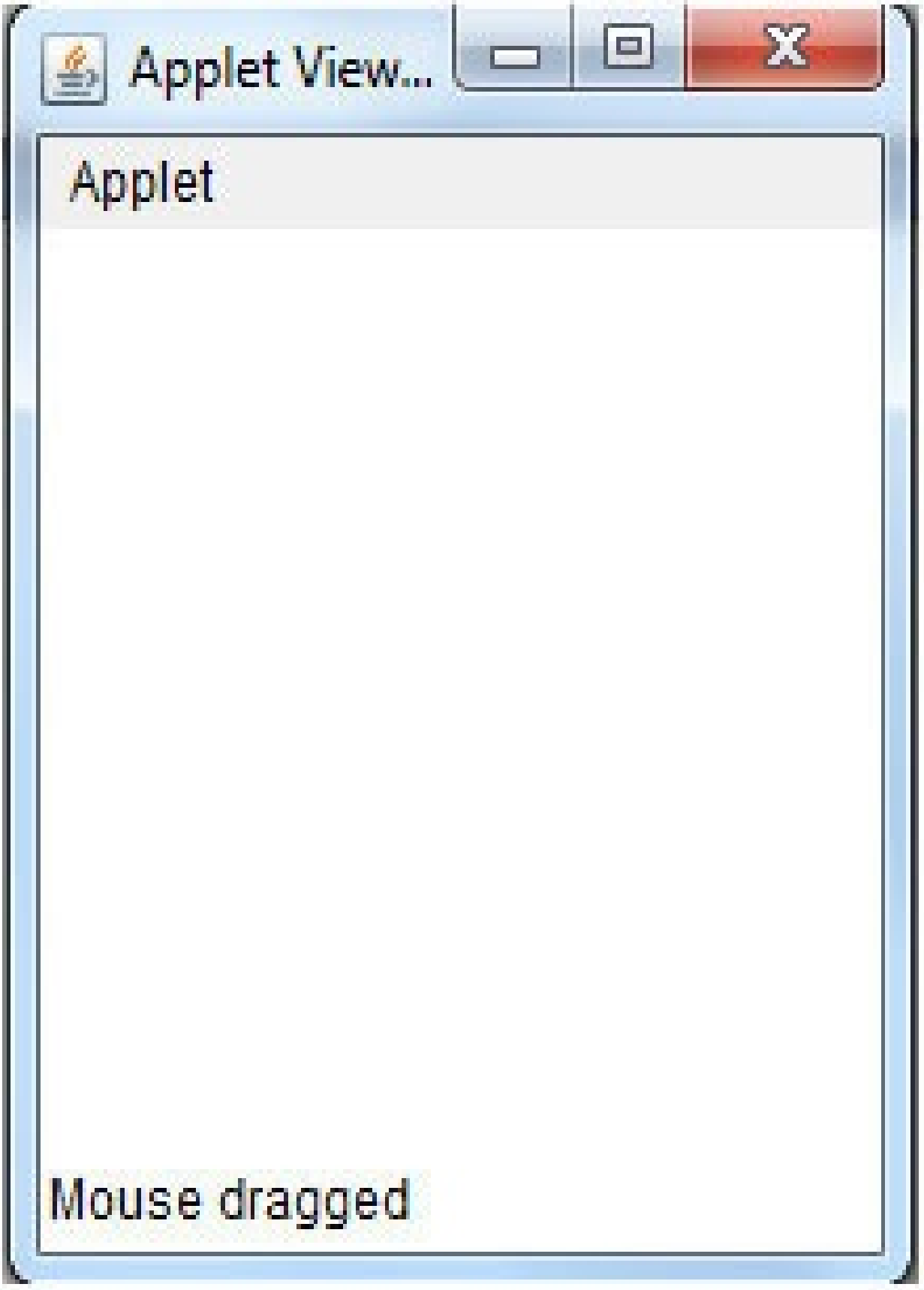

Applet View...
Applet
Mouse dragged

Figure 10.11

Anonymous inner classes

An anonymous inner class is one that is not allocated a name but it can still do its assigned activity and can ease the writing of event handlers. Let us take an example of an applet shown in the following lines of code. The AppletDemo class is created to handle the mouse events:

```
import java.applet.*;

import java.awt.event.*;

/*

<applet code="AppletDemo" width=300 height=300>

</applet>

*/

public class AppletDemo extends Applet

{

public void init()

{

addMouseListener(new MouseAdapter()

{

public void mouseClicked(MouseEvent me)

{
```

```
showStatus("Mouse Clicked");

}

});

}

}
```

The syntax new MouseAdapter( ) { … } indicates to the compiler that the code between the braces defines an anonymous inner class. Moreover, that class extends MouseAdapter. This new class is not named, but it is automatically instantiated when this expression is executed. The objective of this anonymous declaration is that we need not to create a separate class and it solves the many problems for a programmer in a much better way. The following program uses the anonymous inner classes to handle the events.

A GUI based form is created which responds to various events:

```
package book;

import java.awt.*;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import javax.swing.JOptionPane;

public class AnonymousDemo {
```

```
static TextField txtname;
static TextField txtenrolment;
static Button btnsubmit;
static Button btncancel;
String msg;
static String name;
static int en_no;
static Frame f = null;
public static void main(String arg[]) {
f = new Frame("Calculator….");
Label l1 = new Label("Please Fill the Form Correctly");
Label l2 = new Label("Enter Name");
Label l3 = new Label("Enter Enrolment");
l1.setBackground(Color.BLUE);;
txtname= new TextField(20);
txtenrolment= new TextField(20);
btnsubmit= new Button("Submit");
btncancel= new Button("Cancel");
f.setLayout(null);
f.setBackground(Color.CYAN);
f.add(l1).setBounds(150, 50, 340, 40);
```

```
f.add(l2).setBounds(150, 100, 100, 40);
f.add(l3).setBounds(150, 150, 100, 40);
f.add(txtname).setBounds(300, 100, 175, 30);
f.add(txtenrolment).setBounds(300, 150, 175, 30);
f.add(btnsubmit).setBounds(350, 250, 100, 30);
f.add(btncancel).setBounds(200, 250, 100, 30);
txtname.addKeyListener(new KeyAdapter(){
public void keyTyped(KeyEvent arg0)
{
if ((arg0.getSource() == txtname) && (Character.isDigit(arg0.getKeyChar())))
{
JOptionPane.showOptionDialog(
f,"Numbers Not Allowed",
"Enter a valid Name",
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
null,
null,
null);
arg0.setKeyChar((char) KeyEvent.VK_CANCEL);
}
```

```
}

});

txtenrolment.addKeyListener(new KeyAdapter(){

public void keyTyped(KeyEvent arg0)

{

if ((arg0.getSource() == txtenrolment) &&
(Character.isAlphabetic(arg0.getKeyChar())))

{

JOptionPane.showOptionDialog(

f,"Characters Not Allowed",

"Enter a valid Enrolment",

JOptionPane.DEFAULT_OPTION,

JOptionPane.INFORMATION_MESSAGE,

null,

null,

null);

arg0.setKeyChar((char) KeyEvent.VK_CANCEL);

}

}

});

btnsubmit.addMouseListener(new MouseAdapter(){
```

```
@Override
public void mouseClicked(MouseEvent arg0) {
// TODO Auto-generated method stub
name= txtname.getText();
en_no= Integer.parseInt(txtenrolment.getText());
JOptionPane.showOptionDialog(f,"Hello "+ name +
" Your enrolment no "
+ en_no +" has been submitted",
"Form Submission",
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
null,
null,
null);
}
});

f.setSize(500, 500);
f.setVisible(true);
}
}
```

The output is same as in previous program.


Calculator....
Please Fill the Form Correctly
Enter Name
Faheem Bhat
Enter Enrolment
700
Cancel
Submit
Form Submission
Hello Faheem Bhat Your enrolment no 700 has been submitted
OK

Figure 10.12

Assignments

Explain the delegation event model with the help of a Java program that creates a GUI and responds to various controls of GUI?

What is the difference between an event and event listener?

What is the purpose of Adapter classes in event handling?

What is the advantage of using anonymous inner declaration in event handling?

Create a Java program that performs the different operations of a calculator?

Create a GUI with four buttons say PRIME SERIES, FABSERIES, EVEN SERIES, and ODD SERIES. If a user clicks on any of the buttons, it should generate the corresponding series?

CHAPTER 11

Java Database Connectivity

Structure

Introduction

Features of JDBC

Objectives of JDBC

Overview of SQL

JDBC architecture

Implementation of JDBC

Assignments

Introduction

The Java programming language has a lot of potential for the database programming as programmers can write applications in the Java programming language to access whichever database by means of the Structured Query Language (SQL) statements or even special extensions of SQL at the same time following Java language conventions. This chapter explains the key ideas in implementing the JDBC that is the Java database connectivity. The main focus of this chapter is to implement the Java database connectivity by making use of the Java API packages. This chapter starts with the features of JDBC to understand the advantages of using Java for database connectivity. The objectives of JDBC are explored in this chapter. The key for implementation of JDBC is the SQL commands. We must have a proper understanding of those commands then only we can perform various operations like insertion, updation, and creation of tables within a database. This chapter gives an overview of SQL to provide a basic platform for a programmer to go ahead with the JDBC. The five main classes used for implementation are explored and this chapter ends with the practical programming examples.

Features of JDBC

The implementation of database connectivity through Java has many advantages. In addition to the features of Java, the following features are added by making use of the JDBC:

Write once, run anywhere: One of the important feature of Java which is automatically applicable to its database connectivity is platform independent. The explanation that permits Java to resolve both the security and the portability problems is that the output of a Java compiler is not executable code. Rather, it is bytecode. Bytecode is an extremely optimized set of instructions intended to be executed by the Java runtime system. The other important feature of using Java for database connectivity is that we can have multiple clients and multiple servers.

Object-relational mapping: As we already discussed, the Java is a pure object oriented wherein everything is in terms of classes and objects. So is the database connectivity implemented in Java with a package called java.sql, which includes number of classes with number of methods to perform diverse range of functionalities. So the databases are optimized for searching/indexing and objects of different classes are optimized for engineering/flexibility.

Network independence: The other issue that concerns the programmers is whether the database will get connected through different protocols. Java databases works across all the Internet Protocols.

Database independence: Java is not adamant that a specific database will only work say Access instead Java can access any database vendor be it Oracle, Access, or any other database.

Ease of administration: The installation is another issue that may arise for the developers. The Java makes it easy for any clients to install the drivers and we call it as zero-install client.

Objectives of JDBC

Simple and familiar: One of the important features of Java is simple and familiar so is the feature of JDBC. We need not to do anything extraordinary in JDBC, the package that we are going to use is java.sql and we are already familiar with packages in Java. The classes that are present in this package are as simple as any other class in Java.

SQL-level: The SQL which is well established and used universally by the developers is also used in JDBC.

100% pure Java: The Java constructs are used in JDBC and nothing new is incorporated.

High-performance: The performance of JDBC is high as compared to ODBC and other database connectivity.

Leverage existing database technology: As already mentioned, the existing database technologies are used and nothing new is reinvented by the JDBC.

Uses strong, static typing wherever possible.

Uses multiple methods to express multiple functionality.

Overview of SQL

The SQL having standardized syntax for querying (accessing) a relational database, works across all databases so it is independent of the database. JDBC allows us to communicate with databases using SQL. This section briefly explains the SQL, if you have never used SQL before then this section may not be sufficient and if so you are recommended to go through the various books on this topic.

SQL functions fit into two extensive categories:

Data definition language includes commands to create database objects, such as tables, indexes, and views Define access rights to those database objects

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Constraint used to validate data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and thus its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

Table 11.1: SQL data definition commands

Data manipulation language includes commands to insert, update, delete, and retrieve data within database tables:

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values

Table 11.2: SQL data manipulation commands

COMMAND OR OPTION	DESCRIPTION
COMPARISON OPERATORS	
=, <, >, <=, >=, <>	Used in conditional expressions
LOGICAL OPERATORS	
AND/OR/NOT	Used in conditional expressions
SPECIAL OPERATORS	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
AGGREGATE FUNCTIONS	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

Table 11.3: SQL data manipulation commands

Some of the data manipulation commands are as under:

INSERT INTO table (field1, field2) VALUES (value1, value2);

(Inserts a new record into the named table)

UPDATE table SET (field1 = value1, field2 = value2) WHERE condition;

(Changes an existing record or records)

DELETE FROM table WHERE condition;

(Removes all records that match condition)

SELECT field1, field2 FROM table WHERE condition ;

(Retrieves all records that match condition)

The relevant examples of the above statements are in the next section:

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D)	The declaration NUMBER(7,2) indicates numbers that will be stored with two decimal places and may be up to six digits long, including the sign and the decimal place. Examples: 12.32, -134.99.
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER, but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as "Smith" and "Katzenjammer" are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) will let you store characters up to 25 characters long. However, VARCHAR will not leave unused spaces. Oracle users may use VARCHAR2 as well as VARCHAR.
Date	DATE	Stores dates in the Julian date format.

Table 11.4: Common SQL data types

In order to understand the above commands and data types, Let us create a table Employee within the hr database as under:

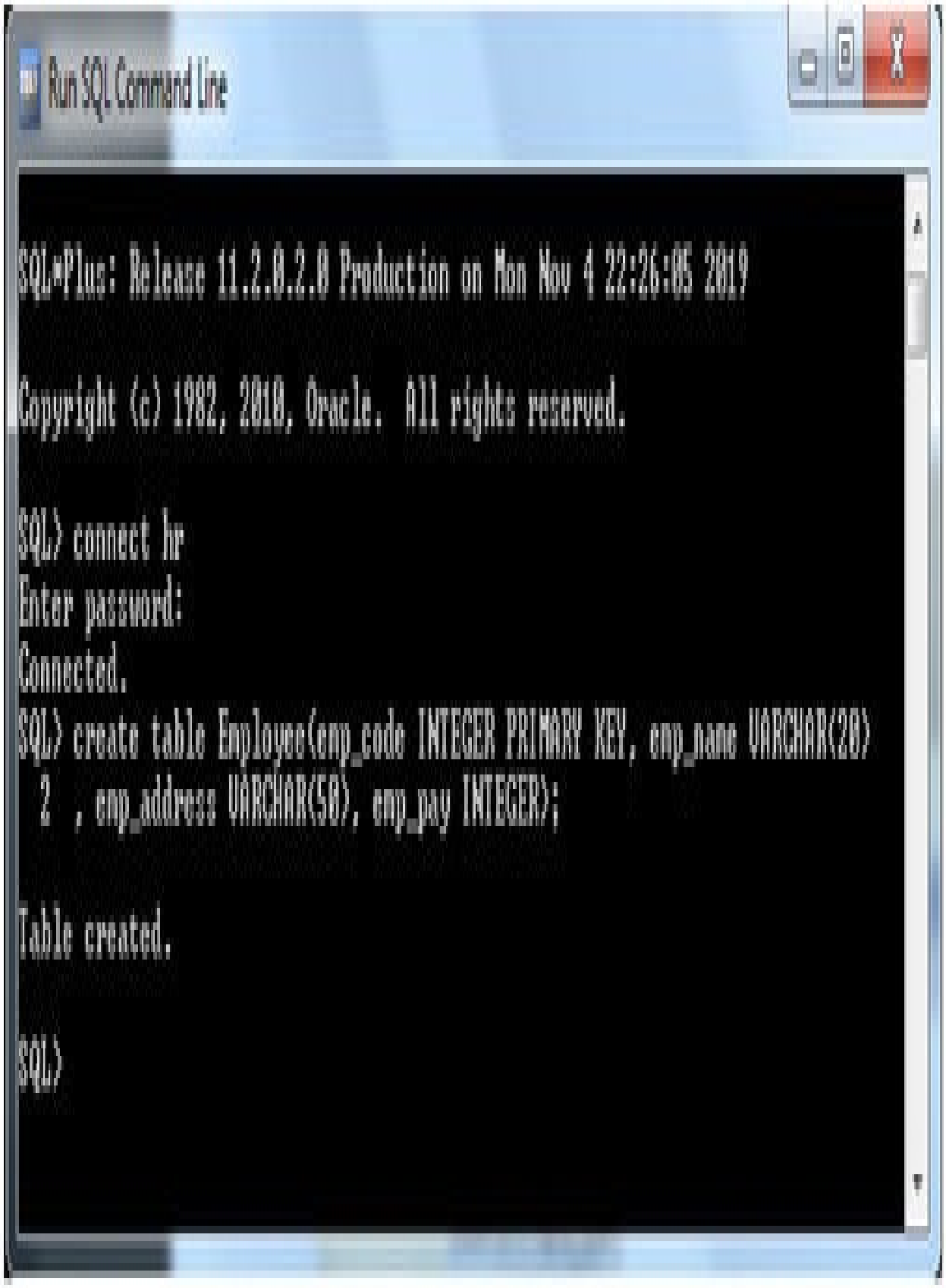

Run SQL Command Line
SQL*Plus: Release 11.2.0.2.0 Production on Mon Nov 4 22:26:05 2019
Copyright (c) 1982, 2010, Oracle. All rights reserved.
SQL> connect hr
Enter password:
Connected.
SQL> create table Employee(emp_code INTEGER PRIMARY KEY, emp_name VARCHAR(20)
2 , emp_address VARCHAR(50), emp_pay INTEGER);
Table created.
SQL>

Figure 11.1: Creation of employee table

The Employee table is created with four columns. The first column is the emp_code with integer data type and is made a primary key, the second attribute is the emp_name with varchar data type, the third and fourth are emp_address and emp_pay with data types as varchar and integer respectively. Now, we can insert the data into the Employee table as under:

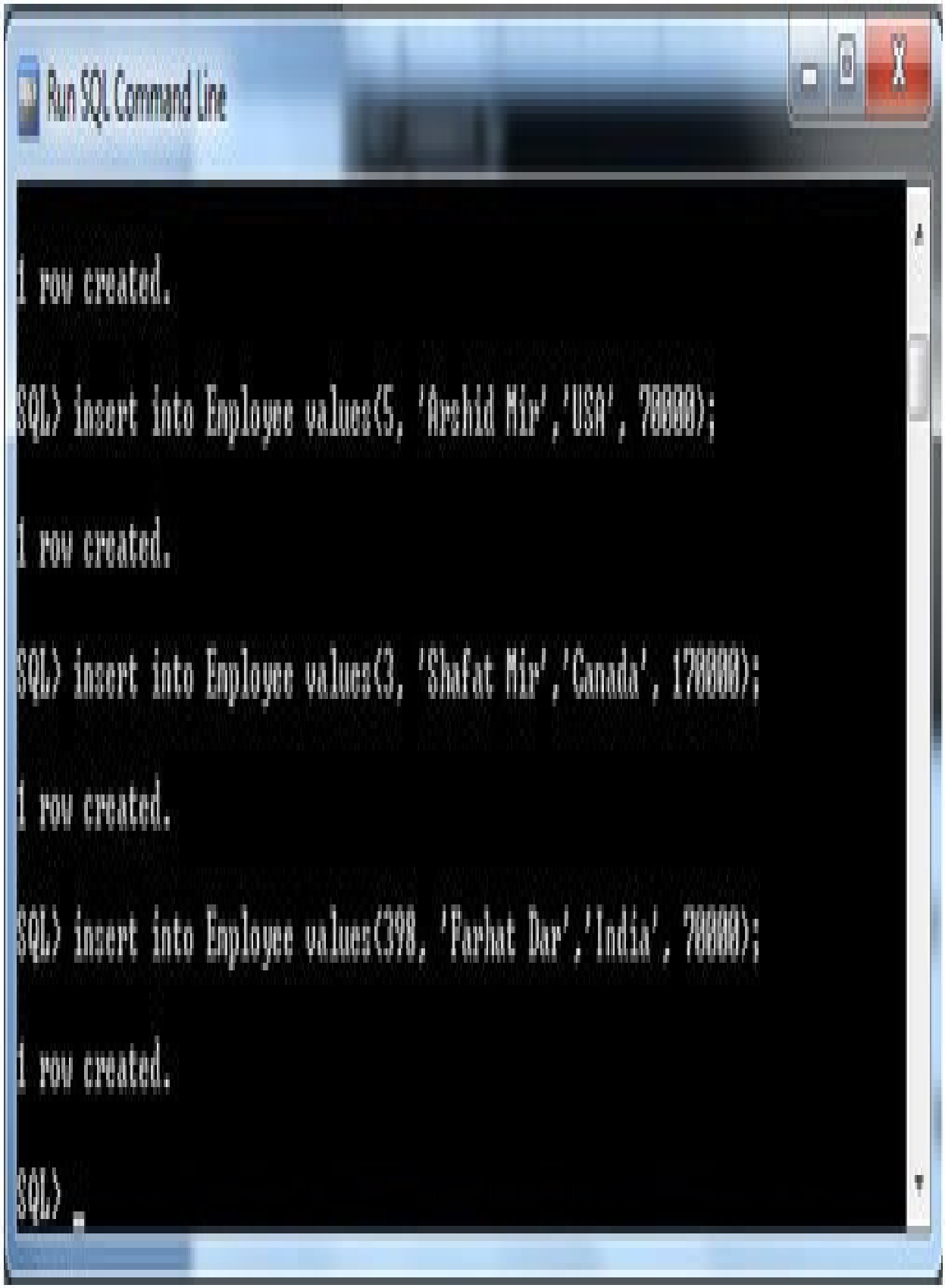

Run SQL Command Line
1 row created.
SQL> insert into Employee values(5, 'Arshid Mir','USA', 70000);
1 row created.
SQL> insert into Employee values(3, 'Shafat Mir','Canada', 170000);
1 row created.
SQL> insert into Employee values(398, 'Farhat Dar','India', 70000);
1 row created.
SQL>

Figure 11.2: Insertion of records into employee table

When entering values into the Employee table, notice that row contents are entered between parentheses, character and date values are entered between apostrophes, numerical entries are not enclosed in apostrophes, attribute entries are separated by commas and a value is required for each column. Use NULL for unknown values.

Now to retrieve the data from the Employee, the select command is used, the three ways are used in the below figure. The first command retrieves all the records from Employee table. The second command only retrieves emp_name and emp_pay and the third statement makes use of where clause to retrieve only those employees whose name is matching the condition. The screenshot of the SQL commands executed are in figure below:

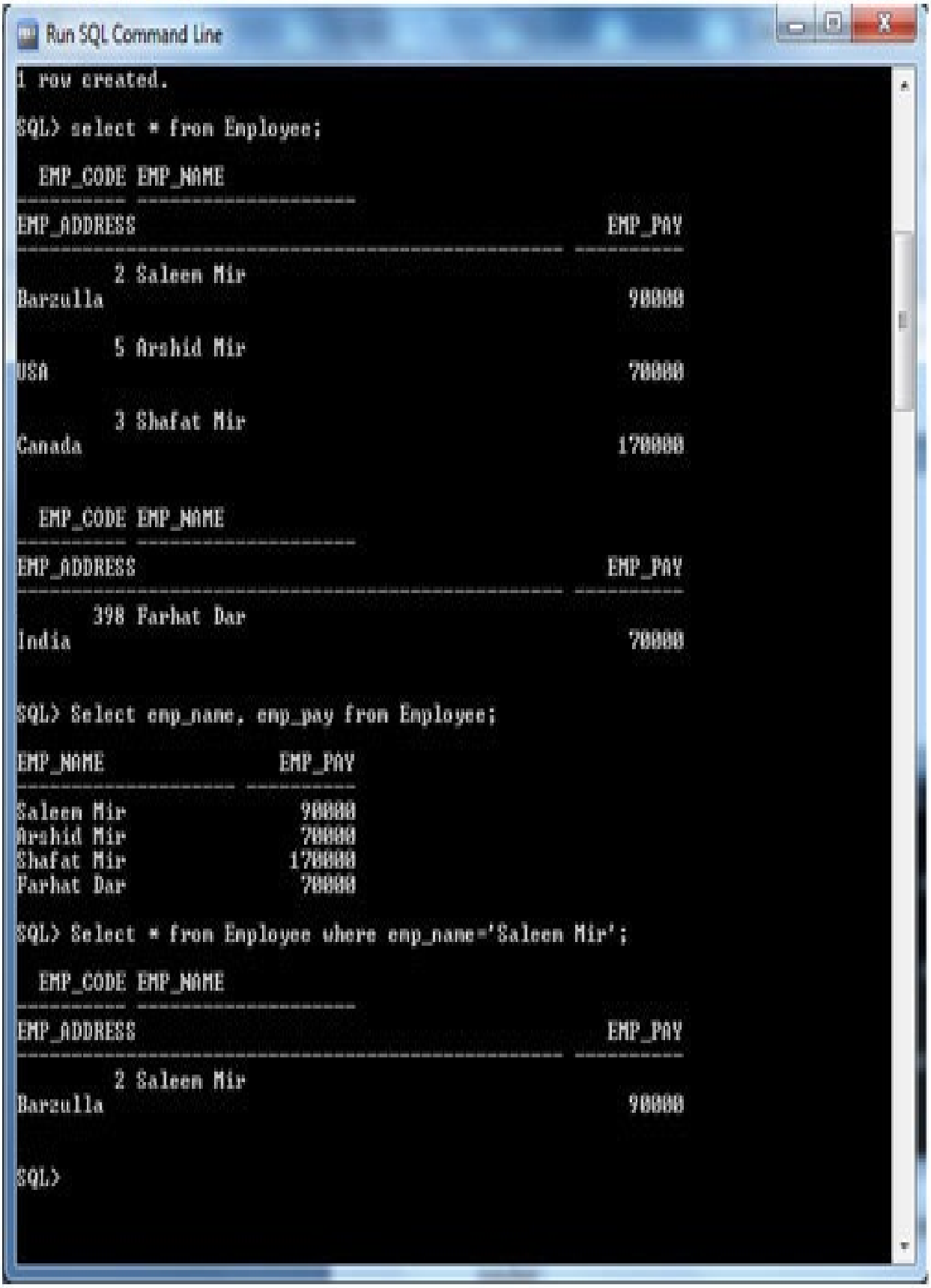

Run SQL Command Line
1 row created.
SQL> select * from Employee;
EMP_CODE EMP_NAME
EMP_ADDRESS EMP_PAY
2 Saleen Mir
Barzulla 90000
5 Arshid Mir
USA 70000
3 Shafat Mir
Canada 170000
EMP_CODE EMP_NAME
EMP_ADDRESS EMP_PAY
398 Farhat Dar
India 70000
SQL> Select emp_name, emp_pay from Employee;
EMP_NAME EMP_PAY
Saleen Mir 90000
Arshid Mir 70000
Shafat Mir 170000
Farhat Dar 70000
SQL> Select * from Employee where emp_name='Saleen Mir';
EMP_CODE EMP_NAME
EMP_ADDRESS EMP_PAY
2 Saleen Mir
Barzulla 90000
SQL>

Figure 11.3: Retrieving data from employee table

Changes made to table contents are not physically saved on disk until, one of the following occurs:

Database is closed

Program is closed

COMMIT command is used

Syntax:

COMMIT [WORK];

Will permanently save any changes made to any table in the database.

ROLLBACK is used to restore database to its previous condition and is only applicable if COMMIT command has not been used to permanently store changes in database.

Syntax:

ROLLBACK;

COMMIT and ROLLBACK only work with data manipulation commands that are used to add, modify, or delete table rows

JDBC architecture

The general architecture of JDBC is depicted in the diagram below. The following sequence of procedures is followed in implementing the database connectivity through Java:

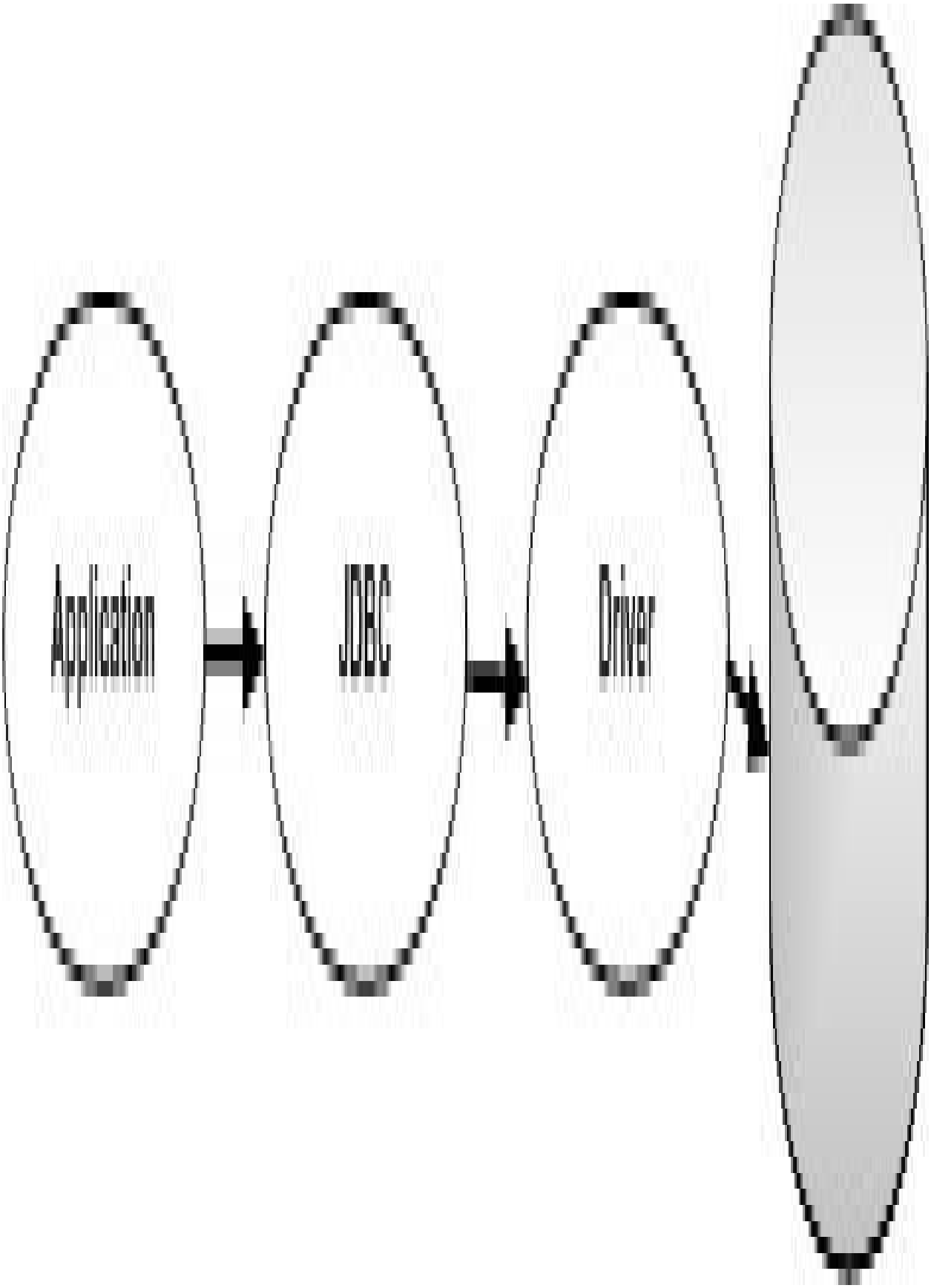

Application
JDBC
Driver

Figure 11.4: JDBC architecture

As we are going to access the database through our Java program, the specialized packages are already in place for the developers to perform multiple functionalities while dealing with databases. One of the important package is the java.sql. So the first step is to load the JDBC library.

In order to deal with a particular database a driver is required. Java can make use of any of the following drivers:

Driver	Description
Type I Drivers:	Uses bridging technology to translate JDBC to ODBC and reli
Type II Drivers	Native API drivers partly written in Java and partly written in
Type III Drivers	It is a pure Java client library and calls middleware server, usu
Type IV Drivers	100% Pure Java -- the Holy Grail, uses Java networking librari

Table 11.5: Type of drivers

So a driver talks to a particular database and can have more than one driver and more than one database.

Implementation of JDBC

The implementation of JDBC is carried out through the following classes and interfaces imported from the java.sql:

DriverManager: Loads, chooses drivers.

Driver: Connects to actual database.

Connection: A series of SQL statements to and from the DB.

Statement: A single SQL statement.

ResultSet: The records returned from a statement.

Driver

This class is useful if we know that we want a specific driver. As such this class is not recommended and we can make use of DriverManager instead of Driver. For example, the below statements explicitly loads a particular driver:

Driver dr = new xyz.may.MyDriver();

Connection c = dr.connect(…);

The various methods of Driver class are as under:

Method of Driver
public abstract java.sql.Connection connect(java.lang.String arg0, java.util.Prope
public abstract boolean acceptsURL(java.lang.String arg0) throws java.sql.SQLI
public abstract java.sql.DriverPropertyInfo[] getPropertyInfo(java.lang.String arg
public abstract int getMajorVersion();
public abstract boolean jdbcCompliant();
public abstract java.util.logging.Logger getParentLogger() throws java.sql.SQLF
public abstract int getMinorVersion();
public abstract boolean acceptsURL(java.lang.String arg0) throws java.sql.SQLI
public abstract int getMajorVersion();
public abstract int getMinorVersion();
public abstract java.sql.Connection connect(java.lang.String arg0, java.util.Prope

public abstract boolean jdbcCompliant();
public abstract java.util.logging.Logger getParentLogger() throws java.sql.SQLF
public abstract java.sql.DriverPropertyInfo[] getPropertyInfo(java.lang.String arg
public abstract java.sql.Connection connect(java.lang.String arg0, java.util.Prope
public abstract boolean acceptsURL(java.lang.String arg0) throws java.sql.SQLF
public abstract java.sql.DriverPropertyInfo[] getPropertyInfo(java.lang.String arg
public abstract int getMajorVersion();

Table 11.6: Methods of Driver

DriverManager

This connects to given JDBC URL with given user name and password. Throws java.sql.SQLException. Returns a Connection object. For example, the following lines of code establish a connection with Oracle database and if successful a Connection object is returned. We must enclose these lines of code in try and catch block as it may throw SQLException or ClassNotFoundException:

```
String dbURL = "jdbc:oracle:thin:@localhost:1521:xe";

String username ="hr";

String password ="hr";

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn = DriverManager.getConnection(dbURL, username,
password);
```

The below table lists the various methods of DriverManager:

Method of DriverManager
private static java.sql.Connection getConnection(java.lang.String arg0, java.util.I
private static void loadInitialDrivers();
private static boolean isDriverAllowed(java.sql.Driver arg0, java.lang.ClassLoad
private static boolean isDriverAllowed(java.sql.Driver arg0, java.lang.Class arg1

public static void println(java.lang.String arg0);
public static void setLogStream(java.io.PrintStream arg0);
public static int getLoginTimeout();
public static java.util.Enumeration getDrivers();
public static synchronized void deregisterDriver(java.sql.Driver arg0) throws jav
public static synchronized void registerDriver(java.sql.Driver arg0, java.sql.Driv
public static java.sql.Driver getDriver(java.lang.String arg0) throws java.sql.SQI
public static java.sql.Connection getConnection(java.lang.String arg0) throws ja
public static java.sql.Connection getConnection(java.lang.String arg0, java.lang.
public static java.sql.Connection getConnection(java.lang.String arg0, java.util.P
public static void setLogWriter(java.io.PrintWriter arg0);
public static java.io.PrintWriter getLogWriter();

Table 11.7: Methods of DriverManager class

Connection

A Connection represents a session with a specific database. Within the context of a Connection, SQL statements are executed and results are returned. Can have multiple connections to a database. Also provides metadata -- information about the database, tables, and fields. Also methods to deal with transactions. In continuation to the lines of code used in the previous section, we can use the Connection object to create a Statement object as showcased below:

```
Statement stmt = conn.createStatement();
```

The following table lists the various methods of Connection:

Method in Connection
public abstract java.sql.Statement createStatement() throws java.sql.SQLExcepti
public abstract java.sql.CallableStatement prepareCall(java.lang.String arg0) thr
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract void setAutoCommit(boolean arg0) throws java.sql.SQLExceptic
public abstract boolean getAutoCommit() throws java.sql.SQLException;
public abstract java.lang.String nativeSQL(java.lang.String arg0) throws java.sql
public abstract void commit() throws java.sql.SQLException;
public abstract void close() throws java.sql.SQLException;
public abstract void rollback() throws java.sql.SQLException;
public abstract java.sql.DatabaseMetaData getMetaData() throws java.sql.SQLE

```
public abstract boolean isClosed() throws java.sql.SQLException;
public abstract boolean isReadOnly() throws java.sql.SQLException;
public abstract java.lang.String getCatalog() throws java.sql.SQLException;
public abstract void setReadOnly(boolean arg0) throws java.sql.SQLException;
public abstract void setCatalog(java.lang.String arg0) throws java.sql.SQLExcep
public abstract java.sql.Statement createStatement(int arg0, int arg1) throws java
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.CallableStatement prepareCall(java.lang.String arg0, int
public abstract java.util.Map getTypeMap() throws java.sql.SQLException;
public abstract void setTypeMap(java.util.Map arg0) throws java.sql.SQLExcept
public abstract void setHoldability(int arg0) throws java.sql.SQLException;
public abstract int getHoldability() throws java.sql.SQLException;
public abstract java.sql.Savepoint setSavepoint() throws java.sql.SQLException;
public abstract java.sql.Savepoint setSavepoint(java.lang.String arg0) throws jav
public abstract void rollback(java.sql.Savepoint arg0) throws java.sql.SQLExcep
public abstract void releaseSavepoint(java.sql.Savepoint arg0) throws java.sql.S
public abstract java.sql.Statement createStatement(int arg0, int arg1, int arg2) thr
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.CallableStatement prepareCall(java.lang.String arg0, int
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.Statement createStatement(int arg0, int arg1) throws java
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.CallableStatement prepareCall(java.lang.String arg0, int
public abstract java.util.Map getTypeMap() throws java.sql.SQLException;
public abstract void setTypeMap(java.util.Map arg0) throws java.sql.SQLExcept
```

public abstract void setHoldability(int arg0) throws java.sql.SQLException;
public abstract int getHoldability() throws java.sql.SQLException;
public abstract java.sql.Savepoint setSavepoint() throws java.sql.SQLException;
public abstract java.sql.Savepoint setSavepoint(java.lang.String arg0) throws jav
public abstract void rollback(java.sql.Savepoint arg0) throws java.sql.SQLExcep
public abstract void releaseSavepoint(java.sql.Savepoint arg0) throws java.sql.S(
public abstract java.sql.Statement createStatement(int arg0, int arg1, int arg2) thr
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.CallableStatement prepareCall(java.lang.String arg0, int
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.PreparedStatement prepareStatement(java.lang.String arg
public abstract java.sql.Statement createStatement(int arg0, int arg1) throws java

Table 11.8: Connection methods

Statement

A Statement object is used for executing a static SQL statement and obtaining the results produced by it. For example, The ResultSet object is retrieved by using the following lines of code:

```
String sql ="SELECT * from Employee";

stmt.executeQuery(sql);

ResultSet rs = stmt.getResultSet();
```

The methods available in Statement are as under:

Methods of Statement
public long executeLargeUpdate(java.lang.String arg0, java.lang.String[] arg1) t
public long executeLargeUpdate(java.lang.String arg0, int[] arg1) throws java.sq
public long executeLargeUpdate(java.lang.String arg0, int arg1) throws java.sql.
public long executeLargeUpdate(java.lang.String arg0) throws java.sql.SQLExc
public long[] executeLargeBatch() throws java.sql.SQLException;
public long getLargeMaxRows() throws java.sql.SQLException;
public void setLargeMaxRows(long arg0) throws java.sql.SQLException;
public long getLargeUpdateCount() throws java.sql.SQLException;
public abstract boolean isCloseOnCompletion() throws java.sql.SQLException;

public abstract void closeOnCompletion() throws java.sql.SQLException;
public abstract boolean isPoolable() throws java.sql.SQLException;
public abstract void setPoolable(boolean arg0) throws java.sql.SQLException;
public abstract boolean isClosed() throws java.sql.SQLException;
public abstract int getResultSetHoldability() throws java.sql.SQLException;
public abstract boolean execute(java.lang.String arg0, java.lang.String[] arg1) th
public abstract boolean execute(java.lang.String arg0, int[] arg1) throws java.sql
public abstract boolean execute(java.lang.String arg0, int arg1) throws java.sql.S
public abstract int executeUpdate(java.lang.String arg0, java.lang.String[] arg1)
public abstract int executeUpdate(java.lang.String arg0, int[] arg1) throws java.s
public abstract int executeUpdate(java.lang.String arg0, int arg1) throws java.sql
public abstract boolean getMoreResults(int arg0) throws java.sql.SQLException;
public abstract java.sql.ResultSet getGeneratedKeys() throws java.sql.SQLExcep
public abstract java.sql.Connection getConnection() throws java.sql.SQLExcepti
public abstract java.sql.ResultSet executeQuery(java.lang.String arg0) throws jav
public abstract int executeUpdate(java.lang.String arg0) throws java.sql.SQLExc
public abstract void close() throws java.sql.SQLException;
public abstract int getMaxFieldSize() throws java.sql.SQLException;
public abstract void setMaxFieldSize(int arg0) throws java.sql.SQLException;
public abstract int getMaxRows() throws java.sql.SQLException;
public abstract void setMaxRows(int arg0) throws java.sql.SQLException;
public abstract void setEscapeProcessing(boolean arg0) throws java.sql.SQLExc
public abstract int getQueryTimeout() throws java.sql.SQLException;
public abstract void cancel() throws java.sql.SQLException;
public abstract java.sql.SQLWarning getWarnings() throws java.sql.SQLExcepti
public abstract void clearWarnings() throws java.sql.SQLException;

public abstract void setCursorName(java.lang.String arg0) throws java.sql.SQLE
public abstract boolean execute(java.lang.String arg0) throws java.sql.SQLExcep
public abstract java.sql.ResultSet getResultSet() throws java.sql.SQLException;
public abstract int getUpdateCount() throws java.sql.SQLException;
public abstract boolean getMoreResults() throws java.sql.SQLException;
public abstract void setFetchDirection(int arg0) throws java.sql.SQLException;
public abstract int getFetchDirection() throws java.sql.SQLException;
public abstract void setFetchSize(int arg0) throws java.sql.SQLException;
public abstract int getFetchSize() throws java.sql.SQLException;
public abstract int getResultSetConcurrency() throws java.sql.SQLException;
public abstract int getResultSetType() throws java.sql.SQLException;

Table 11.9: Statement methods

ResultSet

A ResultSet provides access to a table of data generated by executing a Statement. Only one ResultSet per Statement can be open at once. The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The next( ) method moves the cursor to the next row. As in previous section, we used the Statement object to execute a query and obtain a ResultSet object. The following lines of code makes use of the ResultSet object to retrieve the data from the table:

```
ResultSet rs = stmt.getResultSet();

System.out.println("Retrieving the Data from Student table " );

System.out.println("Roll No Name " );

while(rs.next()){

int roll = rs.getInt(1);

String u = rs.getString(2);

System.out.println(roll + "      " +u);

}
```

The following table lists the methods available in ResultSet:

Method	Description

boolean next()	Activates the next row the first
void close()	Disposes of the ResultSet allow
Type getType(int columnIndex)	Returns the given field as the g
Type getType(String columnName)	Returns the given field as the g
int findColumn(String columnName)	Looks up column index given c
String getString(int columnIndex)	Returns the String as the given
boolean getBoolean(int columnIndex)	Returns the Boolean
byte getByte(int columnIndex)	Returns the Byte
short getShort(int columnIndex)	Returns the Short
int getInt(int columnIndex)	Returns the Int
long getLong(int columnIndex)	Returns the Long
float getFloat(int columnIndex)	Returns the Float
double getDouble(int columnIndex)	Returns the Double
Date getDate(int columnIndex)	Returns the Date
Time getTime(int columnIndex)	Returns the Time
Timestamp getTimestamp(int columnIndex)	Returns the Timestamp
String getString(String columnName)	Returns the String identified by
boolean getBoolean(String columnName)	Returns the boolean identified l
byte getByte(String columnName)	Returns the byte identified by t
short getShort(String columnName)	Returns the short identified by
int getInt(String columnName)	Returns the int identified by the
long getLong(String columnName)	Returns the long identified by t
float getFloat(String columnName)	Returns the float identified by t
double getDouble(String columnName)	Returns the double identified b
Date getDate(String columnName)	Returns the Date identified by t
Time getTime(String columnName)	Returns the Time identified by

Timestamp getTimestamp(String columnName)	Returns the Timestamp identifi

Table 11.10: ResultSet methods

The below class diagram explains the use of various classes as described above. We can start with the DriverManager to get the Connection object which can be further used to get the Statement object and finally use this Statement object to get the ResultSet object to retrieve data from the database. To perform various operations on the database, the various functions/methods are provided by each of the class described above:

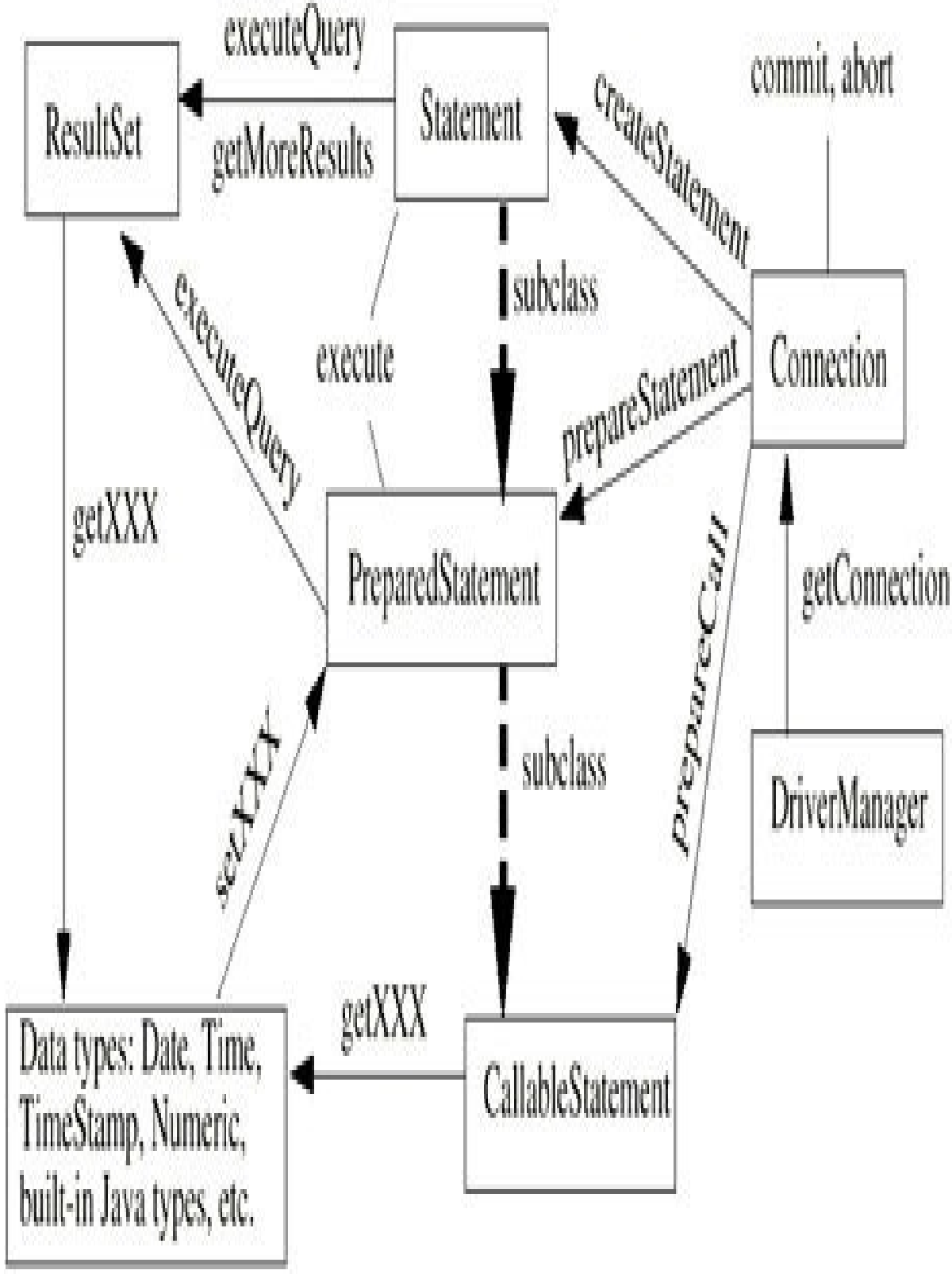

executeQuery
ResultSet
Statement
getMoreResults
commit, abort
createStatement
subclass
executeQuery
execute
Connection
prepareStatement
getXXX
PreparedStatement
getConnection
setXXX
subclass
prepareCall
DriverManager
getXXX
Data types: Date, Time,
TimeStamp, Numeric,
built-in Java types, etc.
CallableStatement

Figure 11.5: Class diagram of JDBC

The following Java program implements the JDBC by retrieving the table data from an Oracle database:

```
package book;

import java.sql.*;

public class MyOracleDatabaseExample {

public static void main(String[] args) throws SQLException,
ClassNotFoundException {

String dbURL = "jdbc:oracle:thin:@localhost:1521:xe";

String username ="hr";

String password ="hr";

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn = DriverManager.getConnection(dbURL, username,
password);

Statement stmt = conn.createStatement();

String sql ="SELECT * from abc";

stmt.executeQuery(sql);

ResultSet rs = stmt.getResultSet();

Boolean r = null;
```

```
System.out.println("Retrieving the Data from Student table " );

System.out.println("Roll No Name " );

while(rs.next()){

int roll = rs.getInt(1);

String u = rs.getString(2);

System.out.println(roll + "     " +u);

}

}

}
```

We will get the following output:

Retrieving the Data from Student table

Roll No Name

10 Muneer

1 Saleem

20 fayaz

The following Java program implements the JDBC by retrieving the table data from an Access database:

```
package book;
import java.sql.*;
class AccessExample
{
public static void main(String a[]) throws SQLException
{
String url = "jdbc:odbc:MyDsn";
Connection con=null;
try {
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
con = DriverManager.getConnection(url);
System.out.println("congrats you are connected");
}
catch (ClassNotFoundException e)
{
System.out.println("Error 1");
}
catch (SQLException e)
{
System.out.println("Error 2");
}
```

```
Statement st = con.createStatement();

ResultSet results = st.executeQuery("select * from Student");

while (results.next()) {

int id = results.getInt(1);

String last = results.getString(2);

System.out.println("" + id + ": " + last);

}

st.close();

con.close();

}

}
```

We will get the following output:

```
congrats you are connected

1: Musadiq

2: Muneer
```

Assignments

Explain the advantages of JDBC over other database programming?

How is the Driver class different from DriverManager class?

List five SQL commands to retrieve data from a table Employee in five different ways?

Write a Java program to insert data into the Employee table discussed in this chapter?

What are the different ways to use the ResultSet object to retrieve the same column?