PG CA S3-04 (DSE)

TWO YEAR MASTER'S DEGREE COURSE

(CBCS in Accordance with NEP 2020)

SEMESTER - III

PROGRAMMING IN JAVA

MScIT

Hiranya Kumar Bhuyan School of Science and Technology

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta, Gauhati University
Prof. (Retd.) Pranhari Talukdar, Gauhati University
Prof. Shikhar Kumar Sarma, Gauhati University
Dr. Jyotiprokash Goswami, Assam Engineering College

Course Coordinators:

Dr. Ridip Dev Choudhury, KKHSOU
Dr. Tapashi Kashyap Das, KKHSOU
Dr. Sruti Sruba Bharali, KKHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
1,2,3,4,5,8	Dr. Utpal Barman, Kaziranga University
6	Dr. Swapnanil Gogoi, GUIDOL
7	Dr. Gautam Chakraborty, NERIM
9, 10, 11	Dr. Gautam Chakraborty, NERIM
12, 13, 14, 15	Ms. Daisy Kalita, USTM

Editorial Team

Content & Language: Dr. Ridip Dev Choudhury, KKHSOU (Unit 1 to 8)
Dr. Utpal Barman, Assam Down Town University (Unit 9 to 15)

Structure, Format & Graphics: Dr. Sruti Sruba Bharali, KKHSOU (Unit 4,5,9 to 15)
Dr. Tapashi Kashyap Das, KKHSOU (Unit 1,2,3,6,7,8)

July, 2024

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-NonCommercial-ShareAlike4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University

Headquarters: Patgaon, Rani Gate, Guwahati 781017; Web : www.kkhsou.in/web/
City Campus : NH 37, Resham Nagar, Khanapara, Guwahati-781022, Assam, India

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY
Programming in Java

DETAILED SYLLABUS

		Page No.
UNIT 1:	**Introduction to Java** An overview of Java, Basic features of Java, JavaEnvironment, Installing the Java SDK, Writing Java Programs	**7-16**
UNIT 2:	**Operators and Assignments** Introduction, Assignment operator, Comparison Operators, Instance of Comparison, Arithmetic operators, Shift operator, Bitwise operator, Logical operator, bitwise operator, conditional operator, casting.	**17-30**
UNIT 3:	**Programming Basic** Java Token and Keywords, Constants, Data types; Declaring a variable, The scope and lifetime of variable, Various Operators, Input/Output statements Decision Making and Control Statements : if statement, If-else, else-if, switch statement; for, while, do-while statements	**31-44**
UNIT 4:	**Class in Java** Class fundamentals: Defining class, Accessing class members, Declaring objects, Passing Arguments to Methods, Returning Multiple Values from methods, Modifiers	**45-57**
UNIT 5:	**OOP in Java** Class fundamentals: Defining class, Accessing class members, Declaring objects, Constructors, copy constructor; Passing Arguments to Methods, modifiers, Inheritance: the super class, Multilevel Inheritance, Final and abstract keyword, Static Members	**58-68**
UNIT 6:	**Arrays, Strings and Vectors** Declaring Arrays, Creating Arrays, Initializing Arrays, Multi-Dimensional Arrays, Strings: string arrays, string methods, String Buffer class, Vectors	**69-108**
UNIT 7:	**Interfaces and Packages** Interfaces: Defining an Interface, Implementing interfaces, Applying Interfaces, Packages: Defining a package, Accessing and Importing Packages	**109-134**
UNIT 8:	**Threads** Introduction, new threads, creating new threads by extending the thread class, creating a thread by implementing Runnable Interface, Threads in the Running State, Sleeping and Interruptions, Signaling with wait, notify	**135-144**

UNIT 9: The java.lang.Math Class ??

Introduction, Methods of the java.lang.Maths: abs, ceil, floor, max, min, random, round, sin, cos, tan, sqrt, exp, log, pow.

UNIT 10: Exception Handling ??

Exception Handling fundamentals, Exception types, Using try and catch, built-In exceptions in Java, User-defined exception.

UNIT 11: File Handling ??

I/O Basics: Streams, The Stream classes, The predefined streams, Reading console input, Writing console output, Reading and writing files

UNIT 12: Introduction to Applets ??

Applets and the World Wide Web, The Applet Class, Applets and HTML , The Life Cycle of an Applet, Using Window Components, Event Handling, Adding Audio and Animation

UNIT 13: AWT and Swings ??

AWT Basics, AWT Components, Event Handling, Application and Menus; Introduction to Swings, Swing Components, Event Handling, Display text and image in a window, Layout manager.

UNIT 14: Introduction to JDBC ??

Basic steps to JDBC, API, JDBC Drivers, Connection Management, JDBC Design Considerations, Two Tier and Three Tier client server model, ResultSet, Prepared statement and callable statement, ResultSet Meta Data Object.

UNIT 15: Database Programming using Java ??

Keeping MySQL or NoSQL as backend and using Java as front end, good GUI should design to insert, update and delete record on a database table.

COURSE INTRODUCTION

Welcome to the “Programming in Java” course. Java is a multi-platform, object-oriented language that can be used as a platform in itself. It has been a popular choice among developers for over two decades, with millions of Java applications in use today. This course is designed to provide provides a comprehensive introduction to Java programming, covering essential concepts and practical skills needed to become proficient in the language.

We will begin with an introduction to Java covering the basics of Java, its key features, and the environment required for Java development. You will learn to install the Java SDK and write Java programs, after understanding the foundation of Java programming including operators, assignments, data types, input/output statements, decision-making and control statements in the next few units. We will then move on to the concept of Object Oriented Programming in Java. The uses of arrays, strings and vectors in Java are explained and the concepts of interfaces, packages and threads are also introduced in this course. The course will delve into the uses of Java.lang.Math class and also cover how the process of exception and file handling mechanism are used in Java. Advanced topics like how to use applets, AWT and swings are also covered in this course. At the end of the course, the concept of JDBC and database programming are also explained. By the end of the course, you will have a solid foundation in Java programming and will be equipped with the skills to develop complex applications and integrate with databases.

Each unit of this course includes some along-side boxes to help you know some of the difficult, unfamiliar terms. Some “EXERCISES” have been included to help you apply your mind to some of the points in the units. You may find some boxes marked with: “LET US KNOW”. These boxes will provide you with some additional interesting and relevant information. Again, you will get “CHECK YOUR PROGRESS” questions. These have been designed to enable you to self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with “ANSWERS TO CHECK YOUR PROGRESS” given at the end of each unit.

UNIT 1: INTRODUCTION TO JAVA

UNIT STRUCTURE

1.1 Learning Objectives

1.2 Introduction

1.3 Introduction to Java

1.4 Basic Features of Java

1.5 Java Environment

1.6 Installing the Java SDK

1.7 Writing Java Programs

1.8 Let Us Sum Up

1.9 Further Reading

1.10 Answer to Check Your Progress

1.11 Model Questions

1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about the Java programming language
- learn the feature of Java
- describe Java environment
- learn to install Java SDK
- write simple programs in Java

1.2 INTRODUCTION

This is an introductory unit on Java programming language. The unit starts with an introduction to Java programming language. In the next section, basic features of Java and Java environment is described. The unit also describes the process of installation of Java SDK. And at the end, the learners will be able to learn how to write a complete simple program in Java programming language.

1.3 INTRODUCTION TO JAVA

Java is a programming language developed by James Gosling at Sun Microsystems Inc. in the year 1995. But later it is owned by Oracle and more than 3 billion devices are using Java. The class-based object-oriented language makes writing, compiling, and debugging programming easy. It is used for modular programs. Java applications are compiled to byte code that can run on any Java Virtual Machine.

After its development, the first public implementation of Java was released by Sun Microsystems in 1996. In the later stage, 2006, the Sun microsystem released the first Java virtual machine as free and open-source software.

Java is a programming language and a platform. Java is a high-level, robust, object-oriented, and secure programming language. Any hardware or software environment in which a program runs are known as a platform. Currently, Java is used in the following devices/applications.

- Acrobat reader, media player, antivirus, etc.
- Railway ticket booking systems such as irctc.co.in.
- Banking applications.
- Mobile
- Internet programming
- E-business
- Robotics
- Games

1.4 BASIC FEATURES OF JAVA

Java is an object-oriented programming language that has the following features.

- Java is a platform-independent language because it is itself a platform. Any hardware or software environment in which a program runs are known as a platform. Since Java has a runtime environment (JRE) and Application Programming Interface (API), it is called a platform.

- Java is more secure than other basic programming languages.
  - It has no explicit pointer.
  - It runs inside a virtual machine sandbox.
- Java is simpler. It is simpler because
  - It is based on C++.
  - It removes explicit pointers, operator overloading, etc from its features.
  - It adds Automatic Garbage Collection for unreferenced objects.
- Java is architecturally neutral. The Java compiler generates the java bytecodes and it does not have any impact on the system architecture. It can interpret with any machine.
- The Java is robust due to the following reasons
  - It has strong memory management.
  - Automatic garbage collection is possible in Java.
  - To get rid of objects issue, Java runs on JVM (Java virtual machine)
  - It eliminates the error-prone situations by emphasizing only compile time error checking and runtime checking.
- Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.



CHECK YOUR PROGRESS

Q 1: State True(T) or False (F):

a. Java is an OOP.

b. Java is more secure.

Q 2: State two features of Java.

.....

.....

Q 3: What is JVM?

.....

.....

1.5 JAVA ENVIRONMENT

Java environment denotes the Java Runtime Environment (JRE). With the help of the Java environment, the Java program runs correctly. The JRE is one of three interrelated components for developing and running Java programs. The other two components of the java environment are as follows:

- The first component is the Java Development Kit or JDK. It is a set of tools for developing Java applications. Developers may choose Java Enterprise Edition (Java EE), Java Special Edition (Java SE), or Java Mobile Edition (Java ME). Every JDK always includes a compatible JRE, because running a Java program is part of the process of developing a Java program. Every JRE includes a default JRE, but developers are free to choose another that meets the specific resource needs of their applications.
- The Java Virtual Machine, or JVM, executes the live Java applications. Every JRE includes a default JRE, but developers are free to choose another that meets the specific resources needs of their applications.

1.6 INSTALLING THE JAVA SDK

In this section, you will learn how to install Java SDK. The following are the steps for the installation process.

- **Verify for Java:** Initially, you have to verify whether Java is installed or not in your system. To perform the verification, type javac in the cmd of your system. If the command is not recognized, then you need to install the Java SDK.
- **Download JDK:** In this step, you can download the Java from https://www.oracle.com/java/technologies/downloads/

Java 18 Java 17

Java SE Development Kit 18.0.2 downloads

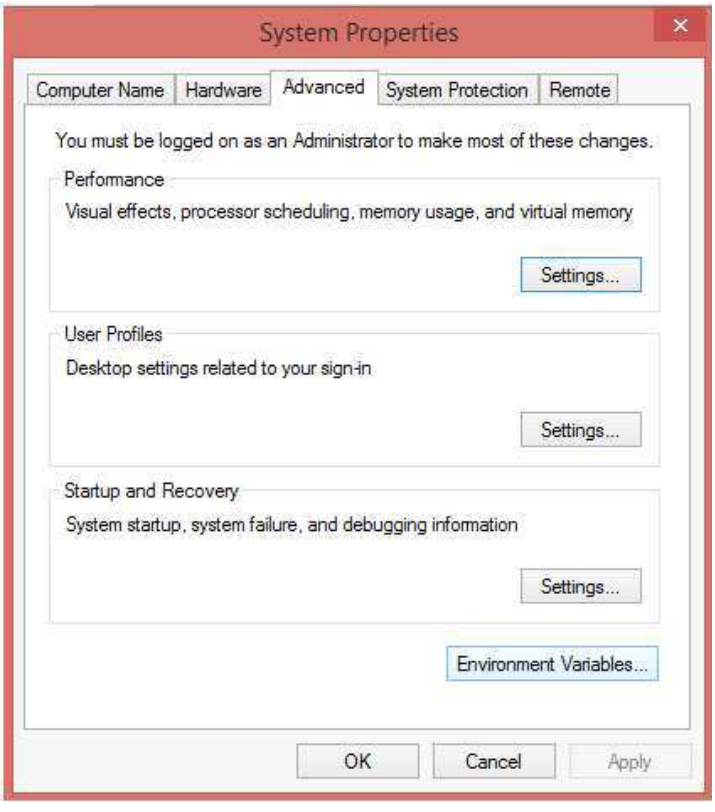
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

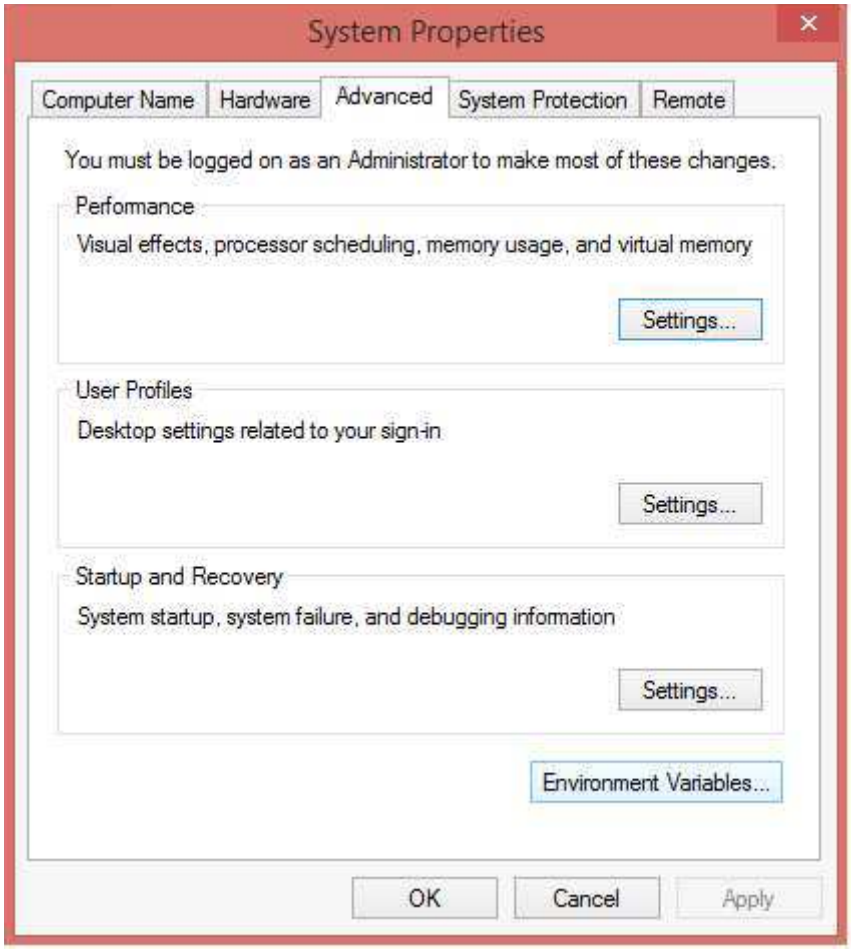
Linux macOS Windows

Product/file description	File size	Download
x64 Compressed Archive	172.79 MB	https://download.oracle.com/java/18/latest/jdk-18_windows-x64_bin.zip (sha256)
x64 Installer	153.37 MB	https://download.oracle.com/java/18/latest/jdk-18_windows-x64_bin.exe (sha256)
x64 MSI Installer	152.25 MB	https://download.oracle.com/java/18/latest/jdk-18_windows-x64_bin.msi (sha256)

- **Install JDK:** Next you have to open the executable file that you have already downloaded and follow the instructions given during the installation. Once it is installed, then close the installation set up.
- **Set the Permanent Path:** For setting the permanent path, you have to open the executable file already downloaded and follow the instructions:
  - Right-click on **"My Computer"**. Then choose **"Properties"** from the options.
  - The following screen will appear. Click on **"Advanced system settings"** to continue.



- Now click on **"Environment Variables"** to continue.



- Enter "path" in the variable name and enter the path to the bin folder inside your JDK in the variable value. Click **OK**.
- Now Java Path has been set up. Open the **command prompt** and type **"javac"** In case you have already opened up the command prompt, I suggest you to close the existing window and reopen it again.

1.7 WRITING JAVA PROGRAMS

In this section, you will learn about the steps of writing a Java program. A Java program is a collection of objects that communicate with each other via java methods.

- **Java Object** - Java Objects are the states and instances of the java class. The object defines the identity of the class. For example, a car is

an object.

- **Java Class** - A class is a frame that defines the behavior/state of the object.
- **Java Methods** - A method is a behavior of an object. A class may have many methods where the data, and logic, are developed.
- **Java Instance Variables** - JAVA object has a unique set of instance variables where values are assigned to these instance variables.

Let us write a Java program. To write a Java program the following steps are required.

- Open **notepad** in your system and start writing your code.
- After writing the code, save the file as **FILENAME.java.** Here, .java is the extension of the java program, and a file name is required to save the file.
- Now, open the command prompt and go to the directory where you have saved your java class.
- Now, in the command prompt type javac **FILENAME.java** and press enter key.
- If there is no error, this will take you to the next line.
- Now type **java FILENAME**
- Then you will be able to see the output of the program.

Let us write a simple Java program.

```
public class FILENAME
{
    public static void main(String [ ]args)
    {
          System.out.println("Hello World");
    }
 }
```

Here, FILENAME is the name of the class and you always save your java program with the same name. The above program will display the word "Hello World". Once you save the file, just follow the following steps to execute the code.

C:\> javac FILENAME.java

C:\> java FILENAME

The output of the program will be: Hello World

To write a Java program, the following points are very important:

- Java is a case-sensitive language.
- For the Class name, the first letter should start with Uppercase.
- All methods' names should start with lower case letters.
- The name of the program file should exactly match the class name.



CHECK YOUR PROGRESS

Q 4: What is the Java environment?

.....

.....

Q 5: What is a Java object?

.....

.....

Q 6: What is Java Class?

.....

.....

Q 7: Write a program in Java to print Hello.

.....

.....



1.8 LET US SUM UP

- Java is a programming language developed by James Gosling at Sun Microsystems Inc in the year 1995. But later it is owned by Oracle and more than 3 billion devices are using Java.
- Java is a programming language and a platform. Java is a high-level, robust, object-oriented, and secure programming language.
- Java is a platform-independent language because it's itself a platform. Any hardware or software environment in which a program runs is known as a platform. Since Java has a runtime environment (JRE) and API, it

is called a platform.

- Java is a more secure programming language.
- It has no explicit pointer.
- It runs inside a virtual machine sandbox
- Java environment denotes the Java Runtime Environment (JRE). With the help of the Java environment, the Java program runs correctly. The JRE is one of three interrelated components for developing and running Java programs.
- Java Objects are the states and instances of the Java class. The object defines the identity of the class. For Example, A car is an object.
- Java Class - A class is a frame that defines the behaviour/state of the object.
- Java Methods - A method is a behaviour of an object. A class may have many methods where the data, and logic, are developed.
- Java Instance Variables - Java object has a unique set of instance variables where values are assigned to these instance variables.



1.9 FURTHER READINGS

1) Schildt, H. (2020). The complete reference Java.



1.10 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: i) True ii) True

Ans to Q No 2: The two features of Java are:

- Java is a platform-independent language because it is itself a platform. Any hardware or software environment in which a program runs is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.
- Java is more secure than other basic programming languages.
- It has no explicit pointer
- It runs inside a virtual machine sandbox

Ans to Q No 3: A Java virtual machine is enables a computer to run Java programs as well as programs written in other languages that are also

compiled to Java bytecode.

Ans to Q No 4: Java environment denotes the Java Runtime Environment (JRE). With the help of the Java environment, the Java program runs correctly. The JRE is one of three interrelated components for developing and running Java programs.

Ans to Q No 5: Java Object - Java Objects are the states and instances of the java class. The object defines the identity of the class. For example, A car is an object.

Ans to Q No 6: A class is a frame that defines the behavior/state of the object.



1.11 MODEL QUESTIONS

Q 1: Explain different features of Java.

Q 2: Write a program in Java to display your name.

Q 3: What are the steps of writing a Java program?

Q 4: How to set up the Java environment.

*** ***** ***

UNIT 2: OPERATORS AND ASSIGNMENTS

UNIT STRUCTURE

2.1 Learning Objectives

2.2 Introduction

2.3 Introduction to Java Operator

2.4 Assignment Operator

2.5 Comparison Operators

2.6 Instance of Comparison

2.7 Arithmetic Operators

2.8 Shift Operator

2.9 Bitwise Operator

2.10 Logical Operator

2.11 Conditional Operator

2.12 Casting

2.13 Let Us Sum Up

2.14 Further Reading

2.15 Answer to Check Your Progress

2.16 Model Questions

2.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about Java operator
- learn the types of operators in Java with examples
- learn about the operator casting

2.2 INTRODUCTION

In the first unit we have already been acquainted with Java environment and various basic features of Java programming language. We have also learnt how to write a simple Java program after installation of Java package. This unit mainly discusses the various operators used in Java programming language. Casting of operators is also necessary to learn which is discussed at the end of this unit.

2.3 INTRODUCTION TO JAVA OPERATOR

Operators are symbols that ask to perform certain specific operations like mathematical or logical manipulation between data items. Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are:

- Arithmetic Operators
- Unary Operators
- Assignment Operator
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise Operators
- Shift Operators
- Instance of operator

2.4 ASSIGNMENT OPERATOR

An assignment operator assigns the value of an operand to a variable. '=' is an assignment operator.

Syntax: variable=value;

Example: x=7;

Program 2.1:

```
// Java code to illustrate "=" operator
import java.io.*;
class Assignment
{
      public static void main(String[] args)
       {
            // Declaring variables
            int num;
            String name;
           // Assigning values
```

```
        num = 11;
        name = "JavaProgramming";
      // Displaying the assigned values
        System.out.println("num is assigned: " + num);
        System.out.println("name is assigned: " +
         name);
    }
}
```

Output: num is assigned: 11

name is assigned: JavaProgramming



CHECK YOUR PROGRESS

Q 1: What is an Assignment operator?

.....

.....

Q 2: X == 7; is it a correct asignment?

.....

.....

2.5 COMPARISON OPERATORS

A comparison operator compares numbers or strings and gives the result of the comparison.

Example:

=	Equal
==	Strictly Equal
\ =	Not Equal
\ ==	Not strictly equal
>	Greater than
<	Less than
> <	Greater than or less than (same as not equal)
> =	Greater than or equal to
\ <	Not less than

< = Less than or equal to

\ > Not greater than

2.6 INSTANCE OF COMPARISON

To check the specific type of instance whether it is a class, subclass, or interface of a given object instance, and return true or false in Java programs is the instance of the comparison operator.

Example:

The Instance of operation is different in different cases.

- **String data type:**

```
class Simple
{
   public static void main(String args[]){
    String str1 = "Educative!";
    if (str1 instanceof String)
       System.out.println("str1 is instance
       of String");
    else
        System.out.println("str1 is NOT instance
        of String");
     }
   }
```

- **Inherited classes:**

```
class Animal { }
class Mammal extends Animal { }
class instanceofTest
{
     public static void main(String[] args)
     {
           Mammal mobj = new Mammal();
           Animal aobj = new Animal();
      // Is `child` class instance of `child` class?
```

```
 if (mobj instanceof Mammal)
      System.out.println("mobj is instance of
      Mammal");
else
      System.out.println("mobj is NOT instance of
      Mammal");
// Is `child` class instance of `parent` class?
if (mobj instanceof Animal)
  System.out.println("mobj is instance of Animal");
else
 System.out.println("mobj is NOT instance of
 Animal");
// Is `parent` class instance of `child` class?
if (aobj instanceof Mammal)
  System.out.println("mobj is instance of Animal");
else
   System.out.println("mobj is NOT instance of
   Animal");
  }
}
```

2.7 ARITHMETIC OPERATORS

The mathematical operations are performed using arithmetic operators. Some of the arithmetic operators are:

- + (addition)

  Syntax:

  num1+num2

  Example:

  num1 = 10, num2 = 20

  sum = num1 + num2 = 30

- \- (subtraction)

  Syntax:

num1 - num2

Example:

```
num1 = 20, num2 = 10
 sub = num1 - num2 = 10
```

- * (multiplication)

  Syntax:

  num1 * num2

  Example:

```
num1 = 20, num2 = 10
mult = num1 * num2 = 200
```

- / (division)

  Syntax:

  num1 / num2

  Example:

```
num1 = 20, num2 = 10
div = num1 / num2 = 2
```

- % (modulus).

  Syntax:

  num1 % num2

  Example:

```
num1 = 5, num2 = 2
mod = num1 % num2 = 1
```



CHECK YOUR PROGRESS

Q 3: Modulus operator, %, can be applied for which purposes?

.....

Q 4: What is/are the highest order precedence operator(s) in Java?

.....

Q 5: Decrement operator ,--, decreases the value of a variable by what number?

.....

2.8 SHIFT OPERATOR

Shifting of bit patterns to the left or right is done using the shift operator. Various types of shift operators are:

- **Signed left shift operator:**

  It moves all the bits to the left by a given number of bits to the left.

  Syntax:

  left_operand << number

  Program 2.2:

```
// Java Program to demonstrate
// Signed Left-Shift Operator
// Importing required classes
import java.io.*;
// Main class
class GFG
{
      // main driver method
      public static void main(String[] args)
      {
            int number = 2;
       // 2 bit left shift operation
            int Ans = number << 2;
            System.out.println(Ans);
      }
}
```

- **Signed right shift operator:**

  It moves all the bits to the right by a given number of bits to the right.

  Syntax:

  left_operand >> number

  Program 2.3

```
  // Java program to demonstrate
  // the Signed right shift operator
```

```
import java.io.*;
class GFG
{
      public static void main (String[] args)
       {
           {
             int number = 8;
             // 2 bit signed right shift
             int Ans = number >> 2;
             System.out.println(Ans);
           }
      }
}
```

- **Unsigned right shift operator:**

  It moves all the bits to the right and the leftmost bit is filled by 0.

  Syntax:

  left_operand >>> number

  Program 2.4:

```
// Java program to demonstrate
// the Unsigned right shift operator
import java.io.*;
class GFG
      {
      public static void main (String[] args)
      {
            byte num1 = 8;
            byte num2 = -8;
            System.out.println(num1 >>> 2);
            System.out.println(num2 >>> 2);
      }
}
```

2.9 BITWISE OPERATOR

The manipulation of discrete bits of a number is done by using a bitwise operator. Various types of bitwise operators are\:

- **Bitwise OR (|)**

  Example:

  a = 5 = 0101 (In Binary)

  b = 7 = 0111 (In Binary)

  Bitwise OR Operation of 5 and 7

  0101

  | 0111

  _____

  0111 = 7 (In decimal)

- **Bitwise AND (&)**

  Example:

  a = 5 = 0101 (In Binary)

  b = 7 = 0111 (In Binary)

  Bitwise AND Operation of 5 and 7

  0101

  & 0111

  _____

  0101 = 5 (In decimal)

- **Bitwise XOR (^)**

  Example:

  a = 5 = 0101 (In Binary)

  b = 7 = 0111 (In Binary)

  Bitwise XOR Operation of 5 and 7

  0101

  ^ 0111

  _____

  0010 = 2 (In decimal)

- **Bitwise Complement (~)**

  Example:

  a = 5 = 0101 (In Binary)

  Bitwise Complement Operation of 5

  ~ 0101

  ————

  1010 = 10 (In decimal)

2.10 LOGICAL OPERATOR

The operators used to make decisions and perform logical operations are called logical operators. Two or more conditions or relational statements are evaluated using logical operators. Various logical operators are:

- AND Operator (&&) - if(a && b) [if true execute else don't]
- OR Operator (||) - if(a || b) [if one of them is true execute else don't]
- NOT Operator (!) - !(a<b) [returns false if a is smaller than b]

  Example:

  x = 5, y = 15, z = 25

  For AND operator:

  Condition 1: z > x

  Condition 2: z > y

  Output: True [Both Conditions are true]

  For OR Operator:

  Condition 1: z > x

  Condition 2: z > y

  Output: True [One of the Condition if true]

  For NOT Operator:

  Condition 1: z > x

  Condition 2: z > y

  Output: False [Because the result was true and NOT operator did it's opposite]

2.11 CONDITIONAL OPERATOR

The operators used to check the conditions of two or more operands and provide the desired result are called conditional operators. Various conditional operators are:

- Conditional AND
- Conditional OR
- Ternary Operator

Program 2.5:

```
public class ConditionalOperatorExample
{
   public static void main(String args[])
   {
      int a=6, b=3, c=9;
      System.out.println(a>b && a>c || b<c);
      System.out.println((a<c || b>c) && a<b);
   }
}
```

Output:

true

false

2.12 CASTING

Converting a data type to another is called type casting. This is applied in situations where we need to store a value of one type into a variable of another type.

Syntax: () is Cast Operator

RequiredDatatype=(TargetType)Variable

Program 2.6:

```
// Java Program to Implement Type Casting of the
Datatype
// Importing input output classes
```

```
import java.io.*;
// Main class
public class XYZ
{
// Main driver method
      public static void main(String[] args)
      {
            // Declaring an Integer datatype
            int a = 6;
            // Casting to Large datatype
            double db = (double)a;
            // Print and display the casted value
            System.out.println(db);
            // Narrow Casting conversion
            int db1 = (int)db;
            // Print an display narrow casted value
            System.out.println(db1);
      }
}
```

Output:

6.0

6



CHECK YOUR PROGRESS

Q 6: What is casting?

Q 7: int a=6, b=3, c=9;

System.out.println(a>b && a>c || b<c);



2.13 LET US SUM UP

- Operators are symbols that ask to perform certain specific operations like mathematical or logical manipulation between data items.
- An assignment operator assigns the value of an operand to a variable.
- A comparison operator compares numbers or strings and gives the

result of comparison.

- To check the specific type of instance whether it is a class, subclass or an interface of a given object instance and return true or false in Java programs is the instance of comparison operator.
- The mathematical operations are performed using arithmetic operators.
- Shifting of bit patterns to the left or right is done using the shift operator. Various types of shift operators are:
  - Signed left shift operator
  - Signed right shift operator
  - Unsigned right shift operator
- The manipulation of discrete bits of a number is done by using a bitwise operator. Various types of bitwise operators are-
  - Bitwise OR (|)
  - Bitwise AND (&)
  - Bitwise XOR (^)
  - Bitwise Complement (~)
- The operators used to make decisions and perform logical operations are called logical operators. Two or more conditions or relational statements are evaluated using logical operators. Various logical operators are:
  - AND Operator (&&) - if(a && b) [if true execute else don't]
  - OR Operator (||) - if(a || b) [if one of them is true execute else don't]
  - NOT Operator (!) - !(a<b) [returns false if a is smaller than b]
- The operators used to check the conditions of two or more operands and provide the desired result are called conditional operators. Various conditional operators are:
  - Conditional AND
  - Conditional OR
  - Ternary Operator
- 10. Converting a data type to another is called type casting. This is applied in situations where we need to store a value of one type into a variable of another type.



2.14 FURTHER READINGS

1) Schildt, H. (2020). The complete reference Java.



2.15 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: An assignment operator assigns the value of an operand to a variable. '=' is an assignment operator.

Ans to Q No 2: No

Ans to Q No 3: Both Integers and floating - point numbers

Ans to Q No 4: () and { }

Ans to Q No 5: 1

Ans to Q No 6: Converting a data type to another is called type casting.

Ans to Q No 7: False



2.16 MODEL QUESTIONS

Q 1: What are the various bitwise operators used in java?

Q 2: Explain shift operators with example.

Q 3: How logical operators are used? Explain with examples.

Q 4: Explain the uses of type casting with examples.

Q 5: Explain the uses of conditional operators with examples.

*** ***** ***

UNIT 3: PROGRAMMING BASIC

UNIT STRUCTURE

3.1 Learning Objectives

3.2 Introduction

3.3 Token, Keywords, and Constant

3.4 Data Types

3.5 Variable Declaration

3.6 The Scope and Lifetime of Variables

3.7 Various Operators

3.8 Input/Output Statements

3.9 Decision Making and Control Statements

3.10 Loop Control Statements

3.11 Let Us Sum Up

3.12 Further Reading

3.13 Answer to Check Your Progress

3.14 Model Questions

3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about Java token, keywords, constant, and about data types
- learn how to declare a Java variable
- describe the input and output system of Java
- learn to use decision and loop statements in Java

3.2 INTRODUCTION

In the previous unit, you have already learned how to run a simple Java program. To write programs in a particular programming language, it is always necessary to learn all the elements of the language.

In this unit, you will be acquainted with the basic elements of Java programing language. The unit starts with the discussion of basic elements like tokens, keywords, constants of Java programming language. It also

includes how to declared variables in Java, different data types, operators, input/output statements, and about scope and lifetime of variables. This unit also discusses the most important decision making and control statements. Loop control statements with their types are also discussed at the end with examples.

3.3 TOKEN, KEYWORDS, AND CONSTANT

Tokens are the smallest discrete unit of a Java program. Tokens are the series of smallest units of the source program which the compiler reads by grouping one character at a time. A Java program is a collection of tokens, comments, and whitespaces. There are five types of tokens:

- Reserved keywords
- Identifiers
- Literals
- Operators
- Separators

Keywords are predefined words having specific meanings for the compiler to understand. Keywords are reserved words defining specific features and they are one of the most essential parts of a language. All keywords must be written in lower-case letters. A basic definition of the Java language is formed using these keywords along with operators and separators according to syntax. There are 50 reserved keywords in Java. A few examples keywords are:

abstract, assert, boolean, break, if,

while, int, float, return, switch, for, etc.

Constants are the fixed entity in the Java program. The value of a constant cannot be changed after it is assigned. Constants are used in programming because it makes the program easy and more understandable to other than the programmer. Several constants supported by Java are:

- Integer constants
- Real constants
- Character constants

- String constants
- Backslash character constants

3.4 DATA TYPES

Data types state the type and size of a variable. Each variable has its data type in Java. There is a wide variety of data types available in Java, making it easy and smooth for a programmer to choose one according to the application and use. Data types are divided into two types:

- Primitive data types: It includes int, long, short, byte, float, double, char and boolean.
- Non-primitive data types: It includes string, array, interface, class, etc.

3.5 VARIABLE DECLARATION

Variables are declared before they are used. The basic structure to declare a variable is:

Syntax: **type identifier;**

where type is the data type and the name of the variable is the identifier. For example: int x;

The same type of variable can be declared by using a comma-separated list:

Syntax: **type identifier1, identifier2, ... identifiers;**

Example: int x, y, z;

Variable can also be declared by initializing with a value:

Syntax: **type identifier = value;**

Example: int x = 5;

3.6 THE SCOPE AND LIFETIME OF VARIABLES

Scope of a variable means where in a program the variable can be accessed. The lifetime of a variable means the period in which the variable exists. Lifetime determines when a variable is created and destroyed.

Java variables are divided into three kinds:

- **instance variables:** Instance variables are declared inside a class and outside all the methods and blocks. The scope of an instance variable is all over the class excluding static methods. The lifetime of an instance variable is upto until the object stays in memory.
- **Class variables:** Class variables are declared inside a class, outside all the blocks, and are marked static. The scope of a class variable is all over the class. The lifetime of a class variable is up until the program ends or until the class is loaded in memory.
- **Local variables:** Local variables are all those variables that are neither instance nor class variables including parameters in a method. The scope of a local variable is within the block where it is declared. The lifetime of a local variable is up until the control leaves the block where it is declared.

3.7 OPERATORS

Operators are symbols that ask to perform certain specific operations like mathematical or logical manipulation between data items. Java operators are classified into the following categories:

- **Arithmetic operators:**

  Examples of arithmetic operators are +(addition), -(subtraction), *(multiplication), /(division) and %(modulus).

- **Arithmetic Assignment operators:** '+=' is an arithmetic assignment operator.

  Syntax: variable operator = expression

  or

  variable = variable operator expression

  Example: x+=5

  Equivalent expression: x=x+5

- **Assignment operators:** '=' is an assignment operator.

  Syntax: variable = value

  Example: x=7;

- **Increment and decrement operators:**

'++' is the increment operator and '--' is the decrement operator. The increment operator increases the operand by 1 and the decrement operator decreases the operand by 1.

- **Relational operators:**

The relationship between two operands is determined by relational operators. '==' is a relational operator which means 'equal to'.

Example: X==Y

- **Logical operators:**

Logical operators check the condition between two or more relational expressions. Types of Logical operators are:

Logical AND &&

Logical OR ||

Logical NOT !

- **Conditional operators:**

Conditional operator is a decision operator. '?:' are conditional operators.

Syntax: test condition? expression1: expression2

Example: max =(x>y)? x:y;

- **Bitwise operators:**

Examples of bitwise operator are: ~, &, |, ^, >>, <<, |=, ^=, >>=, <<=.



CHECK YOUR PROGRESS

Q 1: State True or False:

i. int myNum = 5;

ii. String myText = "Hello";

iii. boolean myBool = true;

Q 2: What is the assignment operator in Java?

.....

Q 3: What are the types of Java datatypes? Give Examples.

.....

3.8 THE SCOPE OF VARIABLES

The input/output control is contained by a class called 'System' in java.lang package. To take any information from the keyboard and display it on the screen, Java uses a Stream object.

`System.in` - refers to the input stream.

`System.out` - refers to the output stream.

The System.out class contains two methods to print the strings and values on screen :

`print(): Syntax: System.out.print(list);`

This prints the value of the variable name specified in the list of output units .

Example :
```
System.out.print("java ");
System.out.print("programming");
```

Output will be: java programming

`println(): Syntax: System.out.println(list);`

This will print the variable name in the list, the cursor is moved by one line and the next statement is printed in the next line.

Example :
```
System.out.println("java");
System.out.println("programming");
```

Output will be: java
programming

3.9 DECISION MAKING AND CONTROL STATEMENTS

Depending on the result of assessment of an expression, a statement or block of statements are executed having the boolean conditions either true or false is called decision control statements.

The following decision control statements are provided by Java:

- if statement
- if-else statement
- else-if statement
- switch statement

- **if statement:** The syntax of if statement is as follows:

```
if(condition)
{
    //block of statements
 }
```

Example:

```
if (20 > 18)
{
  System.out.println("20 is greater than 18");
}
```

- **if-else statement:** The syntax of if-else statement is as follows:

```
if(condition)
{
    //true block statements
}
else
{
    //false block statements
}
```

Example:

```
int a=2,b=4;
if (a>b)
{
     System.out.println("a is greater");
}
else
{
     System.out.println("b is greater");
}
```

- **else-if statement:** The syntax of else-if statement is as follows:

```
 if (condition1)
  {
```

```
// block of statement to be executed if condition1 is true
  }
  else if (condition2)
  {
     // block of statement to be executed if the condition1 is
     //  false and condition2 is true
   }
  else
  {
      // block of statement to be executed if both the
       //conditions are false
   }
```

Example:

```
int time = 20;
if (time < 10)
{
      System.out.println("Good morning.");
}
else if (time < 15)
{
      System.out.println("Good day.");
}
else
 {
           System.out.println("Good evening.");
 }
```

- **switch statement:** The syntax of switch statement is as follows:

```
switch(expression)
{
    case x:  code block
          break;
```

```
    case y: code block;
            break;
            ........
            ........
     default: code block;
}
```

Example:

```
int day = 2;
switch (day)
{
    case 1:
          System.out.println("Monday");
         break;
  case 2:
         System.out.println("Tuesday");
         break;
  case 3:
          System.out.println("Wednesday");
         break;
  case 4:
          System.out.println("Thursday");
         break;
  case 5:
         System.out.println("Friday");
         break;
  case 6:
         System.out.println("Saturday");
        break;
  case 7:
         System.out.println("Sunday");
        break;
}
```

3.10 LOOP CONTROL STATEMENTS

A block of statements is allowed to run for a certain number of times repeatedly. The following loop control statements are provided by Java:

- **for statement**
- **while statement**
- **do-while statement**

➢ **for statement:**

Syntax:

for (statement 1; statement 2; statement 3)
{
// code block to be executed
}

Example:

```
for (int i = 0; i < 10; i++)
{
    System.out.println(i);
}
```

➢ **while statement:**

Syntax: **while (condition)**
{
// code block to be executed
}

Example: int i = 0;

```
while (i < 10)
 {
     System.out.println(i);
     i++;
  }
```

- **do-while statement:**

  Syntax:

  do

  {

  // code block to be executed

  }

  while (condition);

  Example:

```
int i = 0;
do
{
  System.out.println(i);
  i++;
}
while (i < 10);
```



CHECK YOUR PROGRESS

Q 4: What is the output of the following program segment?

```
if (20 > 18)
{
  System.out.println("20 is greater than 18");
}
```

Q 5: What are the primitive data types in Java?

.....

Q 6: What is the size of float and double in Java?

.....

Q 7: What is the output of the following program

```
class increment {
   public static void main(String args[]){
        int g = 3;
        System.out.print(++g * 8);    }
  }
```



3.11 LET US SUM UP

- Tokens are the smallest discrete unit of a Java program.
- Keywords are predefined words having specific meaning for the compiler to understand.
- Constants are the fixed entity in the Java program. The value of a constant cannot be changed after it is being assigned.
- Several constants supported by Java are:
  - Integer constants
  - Real constants
  - Character constants
  - String constants
  - Backslash character constants
- Variables are declared before they are being used. The basic structure to declare a variable is Syntax: type identifier; where type is the data type and identifier is the name of the variable.
- Scope of a variable means where in a program the variable can be accessed. Lifetime of a variable means the period in which the variable exists.
- Operators are symbols that ask to perform certain specific operations like mathematical or logical manipulation between data items.
- The input/output control is contained by a class called 'System' in the java.lang package. To take any information from the keyboard and display it on the screen, Java uses a Stream object.
  - System.in refers to the input stream.
  - System.out refers to the output stream.

  Depending on the result of assessment of an expression a statement or block of statements are executed having the boolean conditions either true or false is called decision control statements.

The following decision control statements are provided by Java :

- if statement
- if-else statement
- else-if statement

- switch statement

A block of statements is allowed to run for a certain number of times repeatedly. The following loop control statements are provided by Java:

- for statement
- while statement
- do-while statement



3.12 FURTHER READINGS

1) Schildt, H. (2020). The complete reference Java.



3.13 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: i) True ii) True iii) True

Ans to Q No 2: Assignment operators:

'=' is an assignment

Syntax: variable=value

Example: x=7;

Ans to Q No 3: Data types state the type and size of a variable. Each and every variable has its own data type in Java. There is a wide variety of data types available in Java , making it easy and smooth for a programmer to choose one according to the application and use. Basically data types are divided into two types :-

Primitive data type : It includes int,long,short,byte,float,double,char and boolean.

Non-primitive data type : It includes string,array,interface,class, etc.

Ans to Q No 4: 20

Ans to Q No 5: There are 8 types of primitive data types- int, char, boolean, byte, long, float, short, double.

Ans to Q No 6: The size of float and double in java is 32 and 64.

Ans to Q No 7: 32



3.14 MODEL QUESTIONS

Q 1: Explain the difference between Java Token, Keywords, and Constant

Q 2: Explain about Java datatypes.

Q 3: Explain the process of variable declaration in Java.

Q 4: What are the different operators in Java? Explain.

Q 5: Write a Program in Java to find the biggest of 2 number.

Q 6: Write a Program in Java to check whether a no is even or odd.

Q 7: Write a Program in java to find the sum of first 10 numbers.

*** ***** ***

UNIT 4: CLASS IN JAVA

UNIT STRUCTURE

4.1 Learning Objectives

4.2 Introduction

4.3 Class Fundamentals

4.4 Accessing Class Members

4.5 Declaring Objects

4.6 Passing Arguments to Method

4.7 Returning Multiple Values from Methods

4.8 Java Modifiers

4.9 Let Us Sum Up

4.10 Further Reading

4.11 Answer to Check Your Progress

4.12 Model Questions

4.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe Java class
- define about the class members
- explain the process of class object declaration
- describe how to return the multiple values from Java methods.

4.2 INTRODUCTION

In the earlier units, we discussed the basic programming concepts of Java. Topics covering basic elements like tokens, keywords, constants of Java programming language were discussed in the previous unit. The previous unit also includes topics like how to declare variables in Java, different data types, operators, input/output statements, and about scope and lifetime of variables. Loop control statements with their types were also discussed in the previous unit.

In this unit, you will be introduced to class fundamentals in Java. Topics like accessing class members and declaring objects are discussed in detail in this unit. In addition to these, concepts related to passing arguments to methods and returning multiple values from methods are also described in this unit. In the next unit, we will explore the topics related to OOP in java.

4.3 CLASS FUNDAMENTALS

A Java program is represented by a class that contains a block of codes and statements. It is a user-defined data type with syntax to define its properties.

You can define the Java class as follows:

Syntax :

```
class class-name
{
        datatype      variable_1;
        datatype instance-variable_2;
        -
        -
        datatype   variable_n;
        datatype method_name1(parameter-list)
        {
        //method body
        }
        -
        -
        datatype method_name(parameter-list)
        {
        //method body
        }
}
```

Instance variables are data or variables defined within a class. Methods contain the codes. Members of the class are the collection of

methods and variables present in a class.

Example:

```
class shelf
{
      double width, height, depth;
      double volume()
      {
            return width*height*depth;
      }
}
```

4.4 ACCESSING CLASS MEMBERS

In Java, all class members are accessed by the "dot" operator.

The syntax for accessing the instance variable of the class is :

object name.variable name;

where object name is the object and variable name is the instance variable.

Example:

```
rectangle2.length= 10;
rectangle2.width= 8;
```

The syntax for accessing instance methods of the class is :

objectname.methodname(parameter-list);

where, objectname is the object, methodname is the instance method and parameter-list is a list of comma separated values.

Example:

```
void getdata(int a, int b) {
      length=a;
      width=b;
}
rect1.getdata(8,10);
```

4.5 DECLARING OBJECTS

An object is declared by defining a variable with its data type and name.

Syntax: Class ClassName;

ClassName object = new ClassName();

Example:
```
Square squ1;
squ1=new Square();
```

4.6 PASSING ARGUMENTS TO METHODS

Arguments are passed to the method through two categories:

- **Call by value:** The value of an augment is copied to the formal parameter of the sub-routine. The sub-routine does not affect the argument used to call it if any changes are made to the parameter.

Example:

```
class test
{
      void meth(int x, int y)
      {
            x*=2;
            y/=2;
      }
}
class abc
{
      public static void main(String[]args)
      {
            test  obj=new test();
            int i-4,j=7;
            system.out.println("Before
             calling  i="+i+"j="+j);
            obj.meth(i,j);
            system.out.println("After
            calling i="+i+"j="+j");
      }
      }
```

Output: Before calling i=4 j=7

After calling i=4 j=7

Call by reference: A reference to an argument is passed to the parameter. Any changes made to the parameter affect the argument used to call it because this reference is used to access the actual argument identified in the call.

Example:

```
class test
{
      int a,b;
      test (int i,int j)
      {
            a=i;
            b=j;
      }
      void meth(test ox)
      {
            ox.a*2;
            ox.b/2;
      }
}
class abc
{
    public static void main(String[]args)
   {
      test obj=new test (4,5);
      system.out.println("Before Calling a= "+
      obj.a+ "b="+ obj.b );
      obj.meth(obj);
       system.out.println("After calling a="obj.a
      + "b="+ obj.b" );
   }
}
```

Output: Before calling a=4 b=5
After calling a=8 b=2

4.7 RETURNING MULTIPLE VALUES FROM METHODS

In Java, the methods can return only one value at a time. Since the returning of multiple values from a method is not possible we use alternatives to return multiple values in Java. Five ways to return multiple values from methods are:

- **Using a POJO class instance:** Using a POJO class instance means to return an instance of a class consisting of all fields to be returned. It is the most common method used to return multiple values. This method provides type-safety and also improves the readability of the program. This class can be a Plain Old Java Object (POJO) with public member variables and a public constructor to begin the fields with corresponding values or a JavaBean for their private member variables.

  Example:

```
// POJO for grouping multiple fields
final class People
{
    public String name;
    public int age;
    public char gender;
  public Person(String name, int age, char gender)
   {
        this.name = name;
        this.age = age;
        this.gender = gender;
  }
}
class Main
{
    public static Person getDetails()
```

```
{
    // return person details from the method
    String name = "Jane";
    int age = 12;
    char gender = 'F';
    return new Person(name, age, gender);
}
 // Return multiple values from a method in Java
public static void main(String[] args)
{
   Person person = getDetails();
   System.out.println("Name is " + person.name);
   System.out.println("Age is " + person.age);
  System.out.println("Gender is " + person.gender);
}
}
```

Output: Name is Jane

Age is 12

Gender is F

- **Using javafx.util.Pair class:** The Pair class included in the javafx.util package representing the name-value pairs can be used for returning multiple values from a method from Java 8 onwards.

  Example:

```
import javafx.util.Pair;
class Main
{
    public static Pair[] getDetails()
    {
    return new Pair[] { new Pair<>("name", "Jane"),
    new Pair<>("age", 12),
    new Pair<>("gender", 'F') };
    }
```

```
// Return multiple values from a method in
//Java 8 and above
 public static void main(String[] args)
 {
      Pair[] person = getDetails();
      for (int i = 0; i < person.length; i++)
      {
      System.out.println("{" + person[i].getKey()
       + " : " +
       person[i].getValue() + "}");
      }
    }
}
```

Output: {name : Jane}
{age : 12}
{gender : F}

- **Return an array of specific type or an object array:** Returning an array is the elementary approach to return multiple values but it neither offers type-safety for an object array nor passes field information to the caller.

  Example:

```
import java.util.Arrays;
class Main
{
    public static int[] getDetails()
    {
        int v1 = 35;
        int v2 = 55;
        int v3 = 95;
       eturn new int[] { v1, v2, v3 };
    }
  // Return multiple values from a method in Java
   public static void main(String[] args)
```

```
    {
          int[] ints = getDetails();
        System.out.println(Arrays.toString(ints));
    }
}
```

Output: [35, 55, 95]

- **Return a collection:** Instead of returning an array a Collection can be returned, like a List.

  Example:

```
import java.util.Arrays;
import java.util.List;
class Main
{
    public static List<Object> getDetails()
    {
        String name = "Jane";
        int age = 25;
        char gender = 'F';
        long income = 100000;
         return Arrays.asList(name, age,
        gender, income);
    }
   // Return multiple values from a method in Java
    public static void main(String[] args)
    {
        List<Object> person = getDetails();
        System.out.println(person);
    }
}
```

Output: [Jane, 25, F, 100000]

- **Using a delimiter:** A string having all the values separated by a delimiter is returned and after returning the string, it is separated using the same delimiter before the values are used inside the call method.

Example:
```
class Main{
    public static String getDetails() {
        return "Jane" + "~" + "Mike";
    }
   // Return multiple values from a method in Java
    public static void main(String[] args)    {
        String[] name = getDetails().split("~");
        for (String s: name) {
            System.out.println(s);
        }
  }
}
```

Output: Jane
Mike



CHECK YOUR PROGRESS

Q 1: What is the default value of the local variables?

.....

Q 2: What are the various access specifiers in Java?

.....

4.8 JAVA MODIFIERS

While declaring a method in a Java program, there is an optional part in the syntax for modifiers. Modifiers are used in method declaration. There are a number of modifiers, they are:

- **public, protected, default or private :** These four modifiers define the scope,what class can invoke which method. They are also called visibility modifiers. They can be one of these values.
- **static :** This modifier is used for declaring class methods and variables. The method can be invoked on the class without creating an instance of the class.
- **abstract :** The abstract method must be overridden in the subclass

and the class should be extended.

- **final :** An overridden method cannot be there in a subclass.
- **native :** Another language is used to implement the method.
- **synchronized :** Only one thread at a time can access the methods.
- **throws :** This method throws a list of exceptions.



CHECK YOUR PROGRESS

Q 3: What is an object?

.....

Q 4: What is object-oriented paradigm?

.....

Q 5: What is a return a collection?

.....

Q 6: What is POJO class instance?

.....



4.9 LET US SUM UP

- A Java program is represented by a class that contains a block of codes and statements.
- Instance variables are the data or variables defined within a class. Methods contain the codes. Members of class is the collection of methods and variables present in a class.
- In Java, all class members are accessed by the "dot" operator. Syntax for accessing instance variable of the class is : objectname. variablename;
- An object is declared by defining a variable with its data type and name.

  Syntax:

  Class ClassName;

  ClassName object = new ClassName();
- **Call by value:** The value of an argument is copied to the formal parameter of the sub-routine. The sub-routine does not affect the argument used to call it if any changes are made to the parameter.

- **Call by reference:** A reference to an argument is passed to the parameter. Any changes made to the parameter affect the argument used to call it because this reference is used to access the actual argument identified in the call.
- Using a POJO class instance means to return an instance of a class consisting of all fields to be returned.
- The Pair class is included in the javafx.util package representing the name-value pairs can be used for returning multiple values from a method from Java 8 onwards.
- Returning an array is the elementary approach to return multiple values but it neither offers type-safety for an object array nor passes field information to the caller.
- Instead of returning an array, a Collection can be returned, like a List.
  Example: import java.util.Arrays;
  import java.util.List;
- While declaring a method in a Java program, there is an optional part in the syntax for modifiers. Modifiers are used in method declaration.



4.10 FURTHER READINGS

1) Schildt, H., & Coward, D. (2014). Java: the complete reference (p. 1312). New York: McGraw-Hill Education.



4.11 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: The local variables are not initialized to any default value, neither primitives nor object references.

Ans to Q No 2: In Java, access specifiers are the keywords that are used to define the access scope of the method, class, or variable. In Java, there are four access specifiers given below.

- Public The classes, methods, or variables which are defined as public, can be accessed by any class or method.
- Protected can be accessed by the class of the same package, by the sub-class of this class, or within the same class.

- Defaults are accessible within the package only. By default, all the classes, methods, and variables are of default scope.
- Private The private class, methods, or variables defined as private can be accessed within the class only.

Ans to Q No 3: The Object is the real-time entity having some state and behavior. In Java, an Object is an instance of the class having the instance variables as the state of the object and the methods as the behavior of the object. The object of a class can be created by using the new keyword.

Ans to Q No 4: It is a programming paradigm based on objects having data and methods defined in the class to which it belongs. The object-oriented paradigm aims to incorporate the advantages of modularity and reusability. Objects are the instances of classes that interact with one another to design applications and programs.

Ans to Q No 5: Instead of returning an array, a collection can be returned, like a list. Example: import java.util.Arrays;
import java.util.List;

Ans to Q No 6: Using a POJO class instance means to return an instance of a class consisting of all fields to be returned. It is the most common method used to return multiple values. This method provides type-safety and also improves the readability of the program. This class can be a Plain Old Java Object (POJO) with public member variables and a public constructor to begin the fields with corresponding values or a JavaBean for their private member variables.



4.12 MODEL QUESTIONS

Q 1: What is object-oriented paradigm?

Q 2: What is the difference between an object-oriented programming language and object-based programming language?

Q 3: What is the purpose of static methods and variables?

Q 4: What are the advantages of Packages in Java?

Q 5: What if I write static public void instead of the public static void?

*** ***** ***

UNIT 5: OOP IN JAVA

UNIT STRUCTURE

5.1 Learning Objectives

5.2 Introduction

5.3 Java Inheritance

5.4 Java Super-Class

5.5 Multilevel Inheritance

5.6 Final and Abstract Keyword

5.7 Static Members

5.8 Let Us Sum Up

5.9 Further Reading

5.10 Answer to Check Your Progress

5.11 Model Questions

5.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe about Java inheritance.
- explain the Java Super Class
- describe the multilevel Inheritance
- describe about the final and abstract keyword

5.2 INTRODUCTION

In the earlier unit, we have discussed about class fundamentals in Java. Topics covering accessing class members and declaring objects were discussed in the previous unit. The previous unit also included topics like concepts related to passing arguments to methods and returning multiple values from methods. Loop control statements with their types were also discussed in the previous unit.

In this unit, you will be introduced to OOP concepts in Java. Topics like inheritance and Java super class are discussed in detail in this unit. In addition to these, multilevel inheritance, final and abstract keywords in

addition to topics like static members are also described in this unit. In the next unit, we will explore the topics related to arrays, strings and vectors in Java.

5.3 JAVA INHERITANCE

Inheritance is the deriving of a new class from an old class. The old class that is inherited is known as the superclass or base class or parent class and the new class that does the inheriting is called the subclass or derived class or child class. In this mechanism, one Java object acquires all the properties of a parent object.

Types of inheritance are mentioned below.

There are different forms of inheritances:

- **Single inheritances:** Deriving a subclass from only one superclass is called single inheritance.
- **Hierarchical inheritance:** Deriving several subclasses from a single superclass is called hierarchical inheritance.
- **Multilevel inheritance:** Deriving a subclass from another derived subclass is called multilevel inheritance.
- **Hybrid inheritance:** Deriving a class using more than one form of inheritance is called hybrid inheritance.

The following is the procedure for defining a subclass

The syntax defining subclass is -

Syntax :

```
class subclass-name extends superclass-name
{
        variable declaration;
        method declaration;
}
```

The keyword extends denotes that the properties are extended to the subclass name from the superclass name, which means the subclass will now contain the variable and methods of the superclass along with its own.

For example,

```
class SIH extends AICTE_SIH
{
  char team_name[20];
  void display();
}
Another example of Java inheritance is :
Class KKHSOU{
void display()
{
    System.out.println("kkhsou_course");
}
}
class KKHSOU_1 extends KKHSOU{
void display1()
{
    System.out.println("CourseWork_2");}
}
class KKHSOU_2{
public static void main(String args[]){
KKHSOU_1 d=new KKHSOU_1();
d.display1();
d.display();
}}
```

5.4 JAVA SUPER CLASS

The base class from which a subclass is derived is the superclass.

Given an example, the Car class (subclass) inherits the variables and methods from the Vehicle class (superclass) :

```
class Vehicle
 {
    protected String brand = "Ford";   // Vehicle
```

```
attribute
public void honk()    // Vehicle method
 {
    System.out.println("Tuut, tuut!");
 }
}
class Car extends Vehicle
{
   private String modelName = "Mustang";
   // Car attribute
   public static void main(String[] args)
   {
      // Create a myCar object
      Car myCar = new Car();
     // Call the honk() method (from the
     //Vehicle class) on the myCar object
    myCar.honk();
// Display the value of the brand attribute (from
the Vehicle // class) and the value of the modelName
from the Car class
     System.out.println(myCar.brand + " "
     + myCar.modelName);
  }
}
```



CHECK YOUR PROGRESS

Q 1: What is Java inheritance?

.....

Q 2: What keyword is used to inherit a class?

.....

Q 3: A class member declared protected becomes a member of a subclass of which type?

.....

Q 4: Does class B extend A {} is the correct way of inheriting class A by class B?

.....

5.5 JAVA MULTILEVEL INHERITANCE

Deriving a subclass from another derived subclass is called multilevel inheritance. The use of a derived class as a superclass is a common concept in object-oriented programming like Java. A chain of classes is built using this concept.

A derived class with multilevel inheritance is declared as :

```
class A
{
        -----
        -----
}
class B extends A       //First level
{
        -----
        -----
}
class C extends B  // Second level
{
        ------
        ------
}
```

Example:

```
class Shape
{
   public void display()
 {
       System.out.println("Inside display");
   }
}
```

```
class Rectangle extends Shape
{
   public void area()
{
      System.out.println("Inside area");
   }
}
class Cube extends Rectangle
{
     public void volume()
     {
        System.out.println("Inside volume");
     }
}
public class Tester
{
   public static void main(String[] arguments)
   {
      Cube cube = new Cube();
      cube.display();
      cube.area();
      cube.volume();
   }
}
```

5.6 FINAL AND ABSTRACT CLASS

The Final keyword is declared to finalize variables and methods used in the class. A class with the Final keyword is called a Final Class. No other class can inherit a class with the Final keyword. Using the Final keyword means the prevention of inheriting any properties or methods of that class.

Example:

```
// Final class
```

```
final class Super
{
    private int data = 100;
}
public class Sub extends Super
 {
    public static void main(String args[])
    {
    }
}
```

The Abstract keyword is used for abstraction.A class with the Abstract keyword is called Abstract Class. An Abstract class consists of both Abstract and concrete methods.

Example:

```
// An abstract class
// Abstract parent class
abstract class Book
{
   // Abstract method without body
    public abstract void page();
}
  // shayar class extends Book class
public class shayar extends Book
{
  // Declaring the abstract method
    public void page()
    {
        System.out.println("Geek");
    }
   // Driver code
    public static void main(String args[])
    {
        Book obj = new shayar();
        obj.page();
    }
}
```

5.7 STATIC MEMBERS

Members of a class that can be accessed without creating an object are called static members. The Static keyword is used to create a static method. The Static keyword can be used in methods, classes, fields, and blocks of a Java program.

Example:

```
public class MyClass
 {
   public static void sample()
   {
      System.out.println("Hello");
   }
   public static void main(String args[])
   {
      MyClass.sample();
   }
}
```



CHECK YOUR PROGRESS

Q 5: What is multilevel inheritance?

.....

Q 6: What is a static member?

.....

Q 7: What is the abstract keyword?

.....



5.8 LET US SUM UP

- Inheritance is the deriving of a new class from an old class. The old class that is inherited is known as the superclass or base class or parent class and the new class that does the inheriting is called the subclass or derived class or child class.

- There are different forms of inheritances: Single inheritances: Deriving a subclass from only one superclass is called single inheritance. Hierarchical inheritance: Deriving several subclasses from a single superclass is called hierarchical inheritance. Multilevel inheritance: Deriving a subclass from another derived subclass is called multilevel inheritance. Hybrid inheritance: Deriving a class using more than one form of inheritance is called hybrid inheritance.
- The syntax defining subclass is -

  Syntax : class subclass-name extends superclass-name

  {

  variable declaration;

  method declaration;

  }
- The base class from which a subclass is derived is the superclass. Given an example, the Car class (subclass) inherits the variables and methods from the Vehicle class (superclass) :

```
class Vehicle
{
   protected String brand = "Ford";
   // Vehicle attribute
   public void honk()    // Vehicle method
    {
      System.out.println("Tuut, tuut!");
    }
}
```

- Deriving a subclass from another derived subclass is called multilevel inheritance. The use of a derived class as a superclass is a common concept in object-oriented programming like java. A chain of classes is built using this concept.
- The Final keyword is declared to finalize variables and methods used in the class. A class with the Final keyword is called a Final Class. No other class can inherit a class with the Final keyword. Using the Final keyword means the prevention of inheriting any properties or methods

of that class.

- The Abstract keyword is used for abstraction. A class with the Abstract keyword is called Abstract Class. An Abstract class consists of both Abstract and concrete methods.
- Members of a class that can be accessed without creating an object are called static members. The Static keyword is used to create a static method. The Static keyword can be used in methods, classes, fields, and blocks of a Java program.



5.9 FURTHER READINGS

1) Schildt, H., & Coward, D. (2014). Java: the complete reference (p. 1312). New York: McGraw-Hill Education.



5.10 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: Inheritance is the deriving of a new class from an old class. The old class that is inherited is known as the superclass or base class or parent class and the new class that does the inheriting is called the subclass or derived class or child class

Ans to Q No 2: The keyword used is : Extends

Ans to Q No 3: private member

Ans to Q No 4: Yes, this is the correct way.

Ans to Q No 5: Deriving a subclass from another derived subclass is called multilevel inheritance. The use of a derived class as a superclass is a common concept in object-oriented programming like Java. A chain of classes is built using this concept. Not directly through class

Ans to Q No 6: Members of a class that can be accessed without creating an object are called static members.

Ans to Q No 7: The Abstract keyword is used for abstraction. A class with the Abstract keyword is called Abstract Class. An Abstract class consists of both Abstract and concrete methods.



5.11 MODEL QUESTIONS

Q 1: What is Inheritance in Java?

Q 2: Why do we need to use inheritance?

Q 3: What is Is-A relationship in Java?

Q 4: What are superclass and subclass?

Q 5: How is Inheritance implemented/achieved in Java?

Q 6: Write the syntax for creating the subclass of a class.

Q 7: What are the advantages of inheritance in Java?

*** ***** ***

UNIT 6: ARRAYS, STRINGS AND VECTORS

UNIT STRUCTURE

6.1 Learning Objectives

6.2 Introduction

6.3 Declaring Arrays

6.4 Creating Arrays

6.5 Initializing Arrays

6.6 Multi-Dimensional Arrays

- 6.6.1 Two-Dimensional Array Declaration
- 6.6.2 Two-Dimensional Array Creation
- 6.6.3 Two-Dimensional Array Initialization

6.7 Arrays Class

6.8 Strings

- 6.8.1 String Class
- 6.8.2 String Arrays
- 6.8.3 String Methods
- 6.8.4 StringBuffer Class

6.9 Vectors

6.10 Let Us Sum Up

6.11 Further Reading

6.12 Answer to Check Your Progress

6.13 Model Questions

6.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about declaration, creation and initialization of arrays in Java
- learn about multi-dimensional arrays in Java
- use strings in Java
- learn about vectors in Java

6.2 INTRODUCTION

In the previous unit, we have discussed the implementation of class and declaration of object in Java. Concept of constructor is discussed in this section. We have learned how to pass arguments to methods. Finally, implementation of inheritance in Java is also discussed in this unit.

We already know that an array is a data structure containing similar type of data where data are stored in contiguous memory locations. Each data available in an array is termed as element of the array. In this unit, we are going to discuss about the implementation of arrays and strings in Java. We will also learn about vectors in Java.

6.3 DECLARING ARRAYS

In Java programming, arrays are container objects that store data of same types. For example, an integer array can store only integer values. In Java, arrays are dynamically allocated and the size of an array is fixed at the time of its creation. Each element of an array is accessed by an integer value termed as subscript. Subscript is used within bracket after the name of the array. In Java, subscript of an array is started with 0.

The first step to work with array is the declaration of arrays. In this section, we will learn about one-dimensional arrays. Arrays with only one subscript are termed as one-dimensional arrays. The syntax of one-dimensional array declaration in Java is presented as follows;

Data_Type [ ] Array_Name ;

Or

Data_Type Array_Name [ ] ;

In the above syntax, Data_Type is any valid Java data type and Array_Name is the name of the declared array. For example, let us consider the following array declaration statement in Java.

```
int[ ] Arr ;
```

The above statement will declare an integer array where the name of the array is Arr.

6.4 CREATING ARRAYS

In the previous section, we have learnt to declare a one dimensional array in Java. Now in the next step, we have to create arrays. In Java, arrays are created in memory by using 'new' operator. The syntax of using 'new' operator to create a one dimensional array is presented as follows.

Array_Name = new Data_Type [Array_Size] ;

In the above syntax, Array_Size is the value that denotes the maximum number of data which can be stored in the array. Array_Size can be a constant or an integer variable. For example, let us consider the following Java statement.

```
Arr = new int [30] ;
```

The above statement will create an integer array, Arr that can store at most 30 integer numbers.

6.5 INITIALIZING ARRAYS

After creation of an array in a program, values can be inserted into the array depending upon the requirement of the program. In Java, the syntax to insert a value into a one dimensional array is presented as follows.

Array_Name[SubscriptValue] = Data;

The above statement will insert the value, 'Data' into the array, 'Array_Name' at the subscript, 'SubscriptValue'.

The following Java program will demonstrate the insertion of values in a one dimensional array.

Example: Java program to insert integer values in a one dimensional array.

```
import  java.util.*;
class insertData
{
      public static void main(String []args)
      {
         int Arr[ ];  // 1-D array declaration
         System.out.println("Enter array size:");
```

```
Scanner S = new Scanner(System.in);
int arrSize = S.nextInt();
//One-dimensional array creation in memory
 Arr = new int[arrSize];
// Insertion of integer values into the array
System.out.println("Enter array elements:");
for(int i=0;i<arrSize;i++)
{
 System.out.println("Enter "+(i+1)+"th element:");
 int data = S.nextInt();
 Arr[i] = data;
}
System.out.println("Array elements are:");
//Display array elements
for(int i = 0;i<arrSize; i++)
{
    System.out.print("  "+Arr[i]);
}
S.close();
}
}
```

Output of the program:

```
G:\java program>javac insertData.java

G:\java program>java insertData
Enter array size:
5
Enter array elements:
Enter 1th element:
12
Enter 2th element:
5
Enter 3th element:
8
Enter 4th element:
23
Enter 5th element:
87
Array elements are:
  12  5  8  23  87
```

In Java, creation and initialization of a one dimensional array can be performed at the time of its declaration. The syntax for this operation in Java is presented as follows.

Data_Type [ ] Array_Name={ Data1, Data2 , Data3 , Data4,……..};

Or

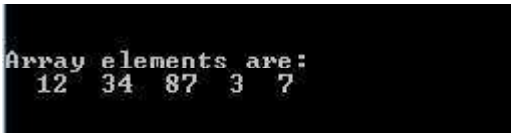
Data_Type Array_Name [ ]={ Data1, Data2, Data3,Data4,……..};

The following Java program will demonstrate the creation and initialization of a one dimensional array at the time of its declaration.

Example: Java program to initialize a one-dimensional array.

```
class initOneDimensionalArray
{
  public static void main(String []args)
   {
      int Arr[ ] = {12,34,87,3,7};
      int N = Arr.length;
     System.out.println("Array elements are:");
     //Display array elements
     for(int i=0;i<N;i++)
       {
        System.out.print("  "+Arr[i]);
       }
    }
}
```

Output of the program:



In the above program, we have used a variable to obtain the size of an array. This variable is 'length'. It is a public final member of array type.

Example: Java program to sort the elements in a one-dimensional integer array using Selection sort algorithm

```
class sortOneDimensionalArray
{
   public static void main(String []args)
   {
     // One-dimensional array initialization
    int Arr[]={5,19,12,34,87,3,7,89,1,33,32,11,};
    int min,i,j,index;
    int N = Arr.length;
    System.out.println("\n\n");
    System.out.println("\n Before sorting
    array elements are:");
     //Display array elements
     for(i=0;i<N;i++)
     {
           System.out.print("  "+Arr[i]);
     }
     // Sorting using Selection sort algorithm
     for(i=0;i<N-1;i++)
     {
         min = Arr[i];
         index = i;
         for(j=i+1;j<N;j++)
         {
             if(Arr[j]<min)
             {
                 min = Arr[j];
                 index=j;
             }
          }
          if(index!=i)
          {
             Arr[index] = Arr[i];
```

```
            Arr[i] = min;
            }
          }
System.out.println("\n After sorting array
elements are:");
 //Display array elements
   for(i = 0;i<N;i++)
   {
       System.out.print("  "+Arr[i]);
   }
    System.out.print("\n\n");
  }
}
```

Output of the program:

```
Before sorting array elements are:
 5  19  12  34  87  3  7  89  1  33  32  11
After sorting array elements are:
 1  3  5  7  11  12  19  32  33  34  87  89
```

6.6 MULTI-DIMENSIONAL ARRAYS

In the previous sections, we have learnt the use of one dimensional arrays in Java. So we already know that, in case of a one dimensional array, only one subscript is available. Now if we declare an array with multiple subscripts than it is termed as multi-dimensional array. So an N-dimensional array possesses N number of subscripts. In Java, a multi-dimensional array can also be referred as arrays of arrays.

6.6.1 Two-Dimensional Array Declaration

Two-dimensional array is an example of multi-dimensional arrays where two subscripts are required to access an array element. Two-dimensional arrays are used to store tables of data. In a two-dimensional array, the first subscript is used to refer the row number and second subscript is used to refer the column number of a table

of data. The syntax to declare a two-dimensional array in Java is presented as follows.

Data_Type [ ][ ] Array_Name ;

Or

Data_Type Array_Name [ ][ ] ;

For example, let us consider the following array declaration statement in Java.

```
int  [ ][ ] ArrTwo ;
```

The above statement will declare a two-dimensional array, 'ArrTwo' that can store integer values.

6.6.2 Two-Dimensional Array Creation

After declaration of a two-dimensional array, it can be created in memory by using the following syntax in Java.

```
Array_Name = new Data_Type [ Array_Size1 ][ Array_Size2 ] ;
```

In the above syntax, Array_Size1 is the maximum number of rows and Array_Size2 is the maximum number of columns that can be possible in the two dimensional array, Array_Name. So, the total number of array elements that can be stored in Array_Name is (Array_Size1 × Array_Size2). For example, let us consider the following Java statement.

```
ArrTwo = new int [10][40] ;
```

The above statement will create a two-dimensional integer array, ArrTwo that can store at most 10 × 40 = 400 integer numbers.

6.6.3 Two-Dimensional Array Initialization

After creation of a two dimensional array, initial values can be stored by using the following syntax in Java.

Array_Name[SubscriptValue1][SubscriptValue2] = Data1;

The above syntax is used to store Data1 into the cell of the two dimensional array, 'Array_Name' whose first subscript is

SubscriptValue1 and second subscript is SubscriptValue2.

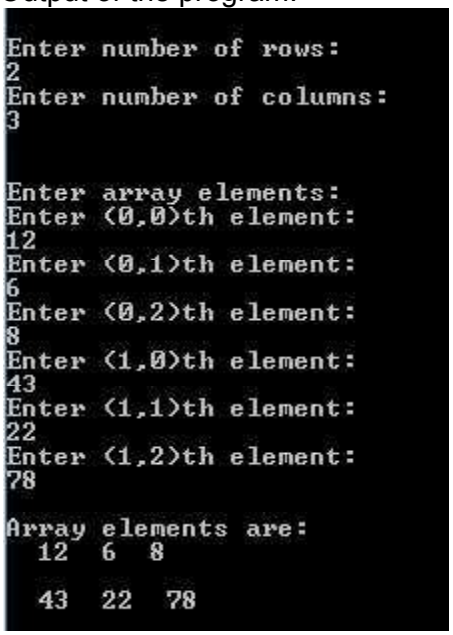
The following Java program will demonstrate the insertion of values in a two dimensional array.

Example: Java program to insert integer data into a two-dimensional array

```
import java.util.*;
class insertDataTwoDArray
{
   public static void main(String []args)
   {
      int arrTwo[][]; //2D-array declaration
     System.out.println("Enter number of rows:");
      Scanner S = new Scanner(System.in);
      int rowNo = S.nextInt();
   System.out.println("Enter number of columns:");
       int colNo = S.nextInt();
      //Two-dimensional array creation in memory
       arrTwo = new int[rowNo][colNo];
  System.out.println("\n\nEnter array elements:");
       //Insertion of integer values into the
       //2D-array
       for(int i=0;i<rowNo;i++)
       {
         for(int j=0;j<colNo;j++)
         {
              System.out.println("Enter
              ("+i+","+j+")th element:");
              int data= S.nextInt();
              arrTwo[i][j]=data;
         }
       }
     System.out.println("\nArray elements are:");
//Display array elements available in the two-
dimensional array
for(int i=0;i<rowNo;i++)
```

```
{
     for(int j=0;j<colNo;j++)
     {
          System.out.print("  "+arrTwo[i][j]);
     }
      System.out.println("\n");
}
    S.close();
   }
}
```

Output of the program:



In Java, creation and initialization of a two-dimensional array can also be performed at the time of its declaration. For example, let us consider the following statements in Java to initialize two dimensional arrays at the time of its declaration.

```
int  ArrTwo1[  ][  ] ={ {2, 5, 3, 9}, {8, 7, 1, 0},
{7, 3, 5, 1}};
int [ ][ ] ArrTwo2={
                    {2, 5, 3, 9},
                    {8, 7, 1, 0},
                    {7, 3, 5, 1}
                };
```

The following Java program will demonstrate the creation

and initialization of a two-dimensional array at the time of its declaration.

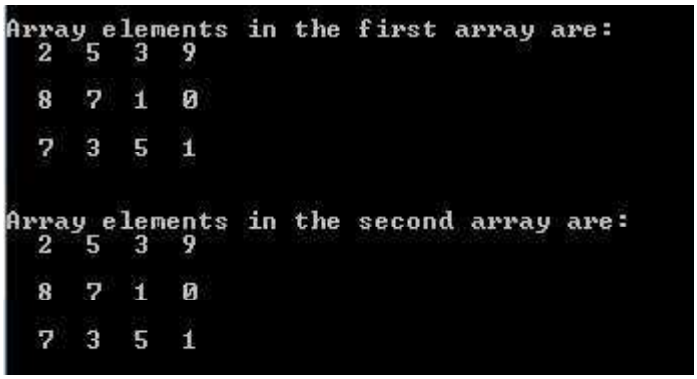
Example: Java program to initialize two-dimensional array

```
class initTwoDArray
{
   public static void main(String []args)
   {
      //Initialization of the first
      //two-dimensional array
       int  arrTwo1[ ][ ] =  { {2, 5, 3, 9},
                               {8, 7, 1, 0},
                               {7, 3, 5, 1}};
//Initialization of the second two-dimensional array
       int  [ ][ ]arrTwo2 =  {
                                   {2, 5, 3, 9},
                                   {8, 7, 1, 0},
                                   {7, 3, 5, 1}
                                };
System.out.println("\nArray elements in the first
array are:");
//Display array elements available in the first
two-dimensional array
   for(int i=0;i<3;i++)
   {
        for(int j=0;j<4;j++)
        {
            System.out.print("  "+arrTwo1[i][j]);
        }
         System.out.println("\n");
    }
 System.out.println("\nArray elements in the second
array are:");
//Display array elements available in the second
//two-dimensional array
for(int i=0;i<3;i++)
 {
```

```
    for(int j=0;j<4;j++)
    {
       System.out.print("  "+arrTwo2[i][j]);
  }
  System.out.println("\n");
 }

}
}
```

Output of the program:



Example: Java program to implement the addition of two two-dimensional arrays

```
import java.util.*;
class addTwoDArray
{
   public static void main(String []args)
   {
    // Two-dimensional array declaration
    int arrTwo1[][],arrTwo2[][],arrResult[][];
    System.out.println("\nEnter number of rows:");
    Scanner S= new Scanner(System.in);
    int rowNo=S.nextInt();
  System.out.println("\nEnter number of columns:");
   int colNo=S.nextInt();
  //First two-dimensional array creation in memory
   arrTwo1 = new int[rowNo][colNo];
  //Second two-dimensional array creation in memory
arrTwo2 = new int[rowNo][colNo];
```

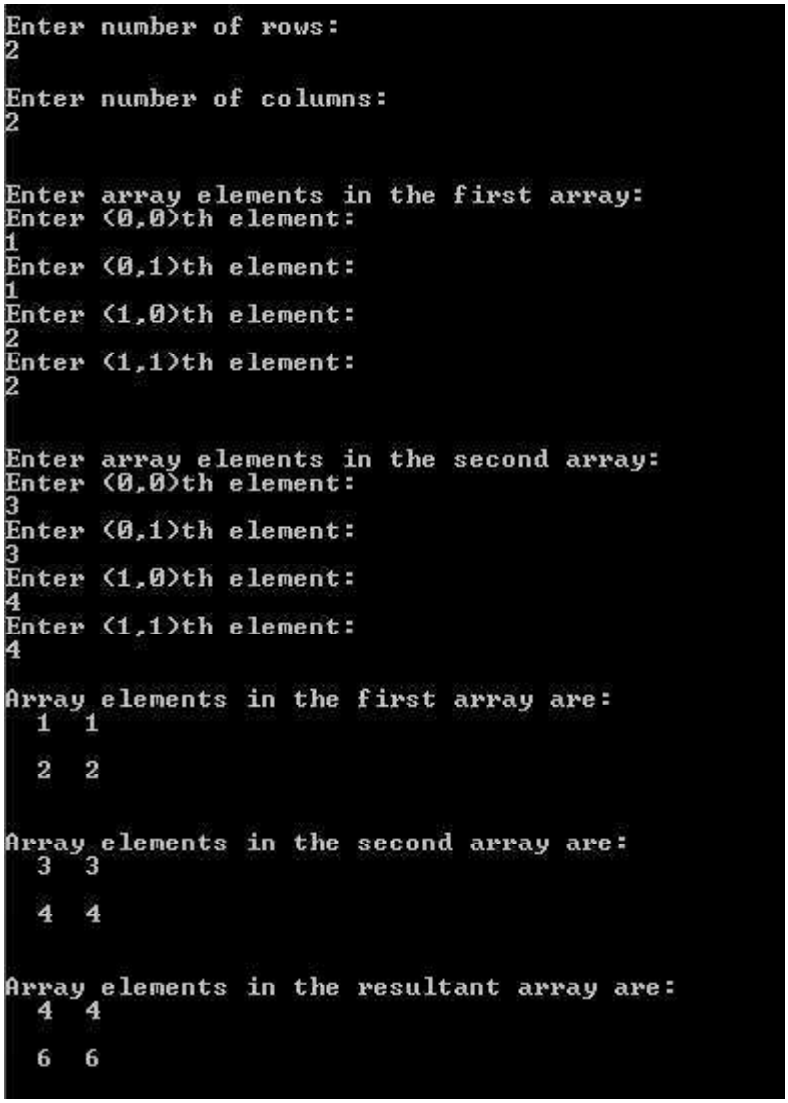
```
//Resultant two-dimensional array creation in memory
 arrResult = new int[rowNo][colNo];
System.out.println("\n\nEnter array elements in the
first array:");
// Insertion of integer values into the first two-
dimensional array
for(int i=0;i<rowNo;i++)
{
     for(int j=0;j<colNo;j++)
     {
        System.out.println("Enter ("+i+","+j+")th
       element:");
       int data= S.nextInt();
       arrTwo1[i][j]=data;
    }
}
System.out.println("\n\nEnter array elements in
the second array:");
// Insertion of integer values into the
//second two-dimensional array
for(int i=0;i<rowNo;i++)
{
     for(int j=0;j<colNo;j++)
     {
        System.out.println("Enter ("+i+","+j+")th
       element:");
       int data= S.nextInt();
       arrTwo2[i][j]=data;
     }
}
// Addition of arrTwo1 and arrTwo2
for(int i=0;i<rowNo;i++)
   {
     for(int j=0;j<colNo;j++)
      {
          arrResult[i][j]=arrTwo1[i][j]+arrTwo2[i][j];
```

```
        }
      }
    System.out.println("\nArray elements in the first
     array are:");
     //Display array elements available in the first
     //two-dimensional array
     for(int i=0;i<rowNo;i++)
      {
         for(int j=0;j<colNo;j++)
         { System.out.print("  "+arrTwo1[i][j]);
            }
           System.out.println("\n");
      }
System.out.println("\nArray elements in the second
array are:");
//Display array elements available in the second
//two-dimensional array
for(int i=0;i<rowNo;i++)
{
   for(int j=0;j<colNo;j++){
   System.out.print("  "+arrTwo2[i][j]);
}
System.out.println("\n");
}
System.out.println("\nArray elements in the
resultant array are:");
//After addition, display array elements available
in the resultant two-dimensional array

 for(int i=0;i<rowNo;i++)
 {
   for(int j=0;j<colNo;j++)
    {
      System.out.print("  "+arrResult[i][j]);
    }
```

```
        System.out.println("\n");
            }
            S.close();
          }
}
```

Output of the program:



6.7 ARRAYS CLASS

'Arrays' is a class available in the Java Collections framework under java. util package. This class contains different static methods to manipulate

arrays in Java. Array of primitive data types or objects may be passed as argument to these methods. These methods may require other arguments also as per the requirement and data type of these arguments must be same with the data type of the array if these are not array index. If some index of the array is required to pass as argument to these methods then it must be an integer value which is less than or equal to the size of the array, otherwise, an exception will be thrown by the corresponding method. 'Arrays' class is inherited from the 'Object' class. Examples of some useful methods provided by Arrays class are presented in the following table.

Function Prototype	Job of the function
static int binarySearch(char chArr[ ], char c)	A particular character value is searched using Binary search algorithm in an array.
static int binarySearch(char chArr[ ], int startIn, int endIn, char c)	A particular character value is searched using Binary search algorithm in a particular range of a character array.
static int compare(char chArr1[ ], char chArr2[ ])	Two character arrays are compared lexicographically.
static int compare(char chArr1[ ], int startIn1, int endIn1, char chArr2[ ] , int startIn2, int endIn2)	Particular ranges of two character arrays are compared lexicographically.
static char chArr1[ ] copyOf(char chArr2[ ], int Ln)	Copies a character array with a particular length into another character array.
static char chArr1[ ] copyOfRange (char chArr2[ ] , int startIn, int endIn)	Copies a particular range of a character array into another character array.
static boolean equals(char chArr1[ ], char chArr2[ ])	Checks whether two characters arrays are equal or not equal. If equal it returns true.

static boolean equals(char chArr1[ ], int startIn1, int endIn1, char chArr2[ ] , int startIn2, int endIn2)	Checks whether particular ranges of two characters arrays are equal or not equal. If equal it returns true.
static void fill(char chArr[ ] , char c)	A particular character is assigned to each array element of a character array.
static void fill(char chArr[ ] , int startIn, int endIn, char c)	A particular character is assigned to each array element of a particular range in a character array.
static int hashCode(char chArr[ ])	A hash code is returned based on the elements of a character array.
static int mismatch (char chArr1[ ] , char chArr2[ ])	Returns the index of the first mismatch between two character arrays. If there is no mismatch found then return -1.
static int mismatch(char chArr1[ ] , int startIn1, int endIn1, char chArr2[ ] , int startIn2, int endIn2)	Checks for the first mismatch between two character arrays in particular ranges and if mismatch is found then return the relative index. If no mismatch is available then return -1.
static void sort (char chArr[ ])	A character is sorted in ascending numerical order.
static void sort(char chArr[ ] , int startIn, int endIn)	A particular range of a character array is sorted in ascending order.
static String toString(char chArr[ ])	String representation for the contents of a character array is returned.

Table 6.1: Examples of Arrays Class Methods

Following Java programs will demonstrate the use of some Arrays class methods.

Example: Java program to copy a character array into another character array and then to sort it using Arrays class methods.

```
class copySortArray{
public static void main(String []args)
{
   int i;
   char [] chArr1 =
   {'A','G','K','M','D','O','F','H','U','B'};
   int N=chArr1.length;
   char [] chArr2 = Arrays.copyOf(chArr1, 10);
   System.out.println("\n\nArray elements in the
   original character array are: \n");
   for(i=0;i<N;i++)
    {
        System.out.print("  "+chArr1[i]);
    }
   System.out.println("\nArray elements in
   the copied character array are: \n");
   for(i=0;i<N;i++)
    {
      System.out.print("  "+chArr2[i]);
    }
   Arrays.sort(chArr2);
   System.out.println("\nAfter sorting,
   array elements in the copied character
   array are:\n");
   for(i=0;i<N;i++)
    {
              System.out.print("  "+chArr2[i]);
    }
```

```
        }
    }
```

Output of the program:

```
Array elements in the original character array are:

  A  G  K  M  D  O  F  H  U  B
Array elements in the copied character array are:

  A  G  K  M  D  O  F  H  U  B
After sorting, array elements in the copied character array are:

  A  B  D  F  G  H  K  M  O  U
```

In the above program, 'char [ ] chArr2 = Arrays.copyOf(chArr1, 10);' is used to copy the character array 'chArr1' into the character array 'chArr2' where the argument represents the number of elements to be copied and that is 10 in this case. Again, 'Arrays.sort(chArr2);' is used to sort the array elements that are available in 'chArr2'.

6.8 STRINGS

String is a collection of character values. So strings can be stored and manipulated using character arrays. In Java, two classes are available to handle strings efficiently. These two classes are String and StringBuffer.

6.8.1 String Class

In Java, strings can be declared as objects of class String. Let us consider the following Java statements to declare and create a string.

```
String string1;
string1 = new String ("Welcome to KKHSOU");
```

In the first statement, string1 is declared as the object of String class. In the second statement, a string, "Welcome to KKHSOU" is created in memory by using new operator. The above statements can also be written as:

```
String string1 = new String ("Welcome to KKHSOU");
```

Some of the commonly used constructors of String class are presented in the following table.

Constructor	Job of the Constructor
String()	This constructor creates String object with empty character sequence.
String(char chArr [ ])	This constructor creates a string with the character sequence that is available in a particular character array.
String(char chArr [ ] , int startInd, int count)	This constructor creates a string with the character sequence that is available in a subarray of a particular character array.
String(String original)	This constructor creates a string object that contains a string which is same with the string object passed as argument.
String(StringBuffer strBuff)	This constructor creates a string object that contains a character sequence which is the content of a particular string buffer.

Table 6.2: Some Commonly Used Constructors of String Class

A simple Java program is presented below to demonstrate the declaration and creation of a string using String class.

Example: Java program to create a string using String class and to display it.

```
class createString
{
   public static void main(String args[])
  {
      // String object declaration
      String string1;
      // Creation of a string in memory
      string1 = new String("Welcome to KKHSOU");
      System.out.println( "\nThe string is:");
      System.out.println(string1);
  }
}
```

Output of the program:

```
The string is:
Welcome to KKHSOU
```

6.8.2 String Arrays

String arrays are the arrays where the array elements are the objects of String class. A string array is used to store multiple strings. Let us consider the following Java statement to create a string array.

String stringArr [ ] = new String [10];

In the above statement, stringArr is a string array that can store at most 10 numbers of strings.

The following Java program demonstrates the creation of a string array.

Example: Java program to create an array of strings and to display it.

```
import java.util.*;
class createStringArray{
public static void main(String args[])
{
    // Creation of string array
    String  stringArr[ ] = new String [ 5 ];
    int i;
    System.out.println("\nEnter five strings:");
    for(i=0;i<5;i++)
    {
   Scanner scr= new Scanner(System.in);
   System.out.print("Enter "+(i+1)+"th string: ");
   stringArr[i]= scr.nextLine(); //Input strings
   }
  System.out.println( "\nEntered strings are:");
  for(i=0;i<5;i++)
  {
        System.out.println(stringArr[i]);
  }
 }
}
```

Output of the program:



In Java, two strings can be concatenated by using '+' operator. The following Java program demonstrates this use of '+'operator.

Example: Java program to concatenate two strings by using '+'operator.

```
class concatenateString
{
      public static void main(String args[])
      {
       // String object declaration
        String string1, string2, string3;
        // Creation of the first string
        string1 = new String ("Welcome to ");
        // Creation of the second string
        string2 = new String ("KKHSOU");
       // Concatenation of string1 and string2
        string3 = string1 + string2;
       System.out.println( "\nThe first
        string is:\n");
       System.out.println(string1);
       System.out.println( "\nThe second
        string is:\n");
       System.out.println(string2);
       System.out.println("\nAfter
      concatenation, the resultant string is:\n");
```

```
        System.out.println(string3);
          }
      }
```

```
The first string is:
Welcome to
The second string is:
KKHSOU
After concatenation,the resultant string is:
Welcome to KKHSOU
```

6.8.3 String Methods

In Java, various methods are provided in String class to work with strings. Some important methods of String class are presented in the following table.

Function Prototype	Job of the function
char charAt(int Ind)	Returns the character available at a particular index in a string.
int compareTo(String string2)	Two strings are compared lexicographically.
int compareToIgnoreCase(String string2)	Two strings are compared lexicographically by ignoring case of the characters.
String concat(String string2)	Concatenates one string to the end of another string.
boolean contains(CharSequence chSeq)	Search for a particular sequence of characters in a string and if found then return true.
boolean contentEquals(CharSequence chSeq)	A particular sequence of characters is compared with a string.
boolean contentEquals(StringBuffer stBuff)	A particular StringBuffer is compared with a string.
boolean endsWith(String su)	Checks whether a string is end with a particular suffix.
boolean equals(String string2)	Returns true if one string is equal to another string.
boolean equalsIgnoreCase(String string2)	Returns true if one string is equal to another string by ignoring case of characters.

void getChars(int start, int end, char chArr [ ] , int chStart)	Characters from a string are copied to a character array.
int hashCode()	Hash code for a string is estimated and returned.
int indexOf(int c)	Returns the index of the first occurrence of a particular character in a string.
int indexOf(int c, int startInd)	Searches a particular character in a string, starting from a particular index and return the index of the first occurrence of the character.
int indexOf(String subStr)	Searches a particular substring within a string and return the index of the first occurrence of the substring.
int indexOf(String subStr, int startInd)	Searches a particular substring within a string, starting from a particular index and return the index of the first occurrence of the substring.
boolean isBlank()	If a string is empty then it returns true.
boolean isEmpty()	If the length of a string is 0 then it returns true.
int lastIndexOf(int c)	Returns the index of a string where a particular character is available for the last time.
int lastIndexOf(int c, int startInd)	Returns the index of a string where a particular character is available for the last time and in this case searching operation is performed in backward direction starting from a particular index.
int lastIndexOf(String subStr)	Returns the index of a string from where a particular substring is available for the last time.
int lastIndexOf(String subStr, int startInd)	Returns the index of a string from where a particular substring is available for the last time and in this case searching operation is performed in backward direction starting from a particular index.
int length()	Returns the length of a string.
String repeat(int num)	Returns a string where the string value is the concatenation of a string repeated N times. N is passed as argument to this method.
String replace(char existChar, char inChar)	Returns a string after replacing all occurrences of an existing character in a string with a new character.

String replaceAll(String reg, String inSubStr)	Replaces all substrings that match a particular regular expression in a string with an input substring.
String replaceFirst(String reg, String inSubStr)	Replaces the first substring that matches a particular regular expression in a string with an input string.
boolean startsWith(String pre)	Returns true if a string starts with a particular prefix.
String strip()	All leading and trailing white spaces of a string are removed and then the string is returned.
Char [ ] toCharArray()	Converts a string to a character array and the new array is returned.
String toLowerCase()	Converts all characters available in a string to lower case.
String toUpperCase()	Converts all characters available in a string to upper case.
static String valueOf(char ch)	String representation of a character is returned.
static String valueOf(char chArr[])	String representation of a character array is returned.

Table 6.3: Some Important Methods of String Class

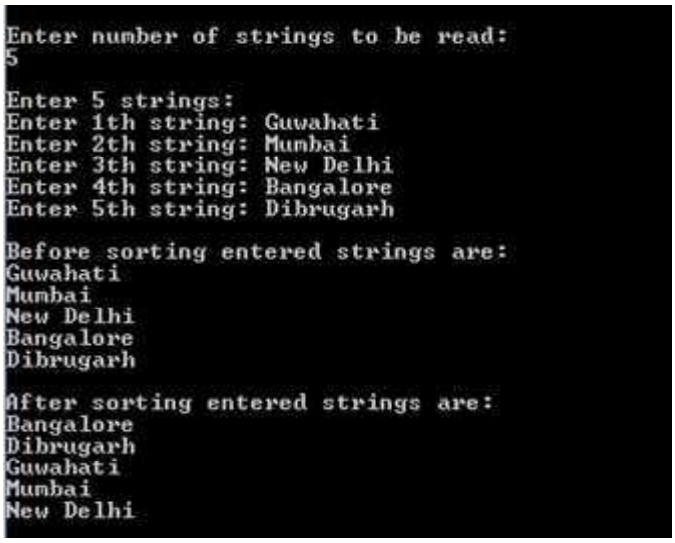
The following program demonstrates the use of some String class methods.

Example: Java program to input strings into an array and to sort it using Bubble sort algorithm.

```
import java.util.*;
class sortStringArray
{
  public static void main(String args[])
  {
  System.out.println("\n\nEnter number of strings
   to be read:");
   Scanner S = new Scanner(System.in);
   int stringNo = S.nextInt();
   String temp=new String();
   // Creation of string array
   String  stringArr[ ] = new String [ stringNo ];
    int i,j;
```

```
System.out.println("\nEnter"+stringNo+" strings:");
for(i=0;i<stringNo;i++){
   Scanner scr= new Scanner(System.in);
  System.out.print("Enter "+(i+1)+"th string: ");
   stringArr[i]= scr.nextLine(); //Input strings
 }
   System.out.println( "\nBefore sorting
   entered strings are:");
for(i=0;i<stringNo;i++){
   System.out.println(stringArr[i]);
}
 // Sorting using Bubble sort algorithm
 for(i=0;i<stringNo-1;i++){
 for(j=0;j<stringNo-i-1;j++)
 {
  if(stringArr[j].compareTo(stringArr[j+1])>1)
   {
        temp = stringArr[j];
       stringArr[j] = stringArr[j+1];
        stringArr[j+1] = temp;
    }
  }
}
System.out.println( "\nAfter sorting entered s
trings are:");
for(i=0;i<stringNo;i++)
{
   System.out.println(stringArr[i]);
}
}
}
```

Output of the program:



In the above program, the String class method, 'compareTo(String string2)' is used to compare two strings.

Example: Java program to input names of some employees and to find out the names which contain 'Kumar' as middle name.

```
import java.util.*;
class findNames
{
      public static void main(String args[]){
      System.out.println("\n\nEnter the number of
       names to be read:";
      Scanner S = new Scanner(System.in);
      int stringNo = S.nextInt();
      String temp = new String();
      // Creation of string array
   String  stringArr[ ] = new String [ stringNo ];
      int i,j;
      System.out.println("\nEnter "+stringNo+"
      Employees'names:");
      for(i=0;i<stringNo;i++)
        {
          Scanner scr= new Scanner(System.in);
          System.out.print("Enter "+(i+1)+"th
```

```
Employee name: ");
stringArr[i]= scr.nextLine(); //Input Employee names
}
System.out.println( "\nNames with middile
name 'Kumar' are:");
for(i=0;i<stringNo;i++)
{
   //Display Employee names which contain 'Kumar'
     as middle name
     temp = stringArr[i].toUpperCase();
     //Convert each character to upper case
      if(temp.contains("KUMAR"))
          System.out.println(stringArr[i]);
  }
}
```

```
Enter the number of names to be read:
5

Enter 5 Employees'names:
Enter 1th Employee name: Bipul Kumar Deka
Enter 2th Employee name: Rinku Moni Nath
Enter 3th Employee name: Ankur Gogoi
Enter 4th Employee name: Rajib Kumar Dubey
Enter 5th Employee name: Pranab Das

Names with middile name 'Kumar' are:
Bipul Kumar Deka
Rajib Kumar Dubey
```

In the above program, the String class method 'toUpperCase()' is used to convert each character of a string into upper case and 'contains(CharSequence chSeq)' is used to check whether the character sequence, 'KUMAR' is available or not available in a string.



CHECK YOUR PROGRESS

Q 1: In Java, subscript of an array is started with _____.

(a) 0

(b) 1

(c) 2

(d) None of the above

Q 2: Which of the following is the correct statement in Java to declare a one-dimensional array?

(a) int arr { };

(b) int [ ] arr;

(c) int ( ) arr;

(d) None of the above

Q 3: Which of the following class is provided in Java to manipulate arrays?

(a) String

(b) Array

(c) Arrays

(d) Both (a) and (b)

Q 4: Which of the following function checks whether two character arrays are equal or not equal?

(a) compare(char chArr1[ ], char chArr2[ ]);

(b) mismatch (char chArr1[ ] , char chArr2[ ]);

(c) equals(char chArr1[ ], char chArr2[ ]);

(d) None of the above

Q 5: Which of the following class is not used to handle strings?

(a) String

(b) StringBuffer

(c) Strings

(d) None of the above

Q 6: Fill in the blanks:

(a) Each element in an array is accessed by an integer value referred as _____.

(b) _____ returns the string representation for the contents of a character array.

(c) hashCode(char chArr[ ]) returns _____ of a character array.

(d) String(char chArr [ ]) constructs a string with _____.

(e) _____returns the character available at a particular index in a string.

6.8.4 StringBuffer Class

One limitation of String class object is that characters of such objects cannot be modified and deleted. Also new characters cannot be added. Some of the String class methods can only return modified new strings but they cannot modify the original strings. So in Java, StringBuffer class is provided to create strings with flexible lengths and the content of these strings can be modified. This means that using StringBuffer class the characters of original strings can be modified and deleted. Also new characters and substrings can be added to the original strings. The constructors of StringBuffer class are presented in the following table.

Constructor	Job of the Constructor
StringBuffer( )	This constructor creates an empty string buffer. In this case the initial capacity of the string buffer is 16 characters.
StringBuffer(int strLen)	This constructor also creates an empty string buffer but in this case the initial capacity is passed as argument.
StringBuffer(String initString)	This constructor creates a string buffer with the characters that are available in a particular string.
StringBuffer(CharSequence chSeq)	This constructor creates a string buffer with the characters that are available in a particular CharSequence.

Table 6.4: Constructors of StringBuffer Class

Some important methods of StringBuffer class are presented in the following table:

Function Prototype	Job of the function
StringBuffer append(char ch)	Appends a particular character to a string that is stored in a StringBuffer object.

StringBuffer append(char chArr [ ])	Appends a particular character array to a string that is stored in a StringBuffer object.
StringBuffer append(String string2)	Appends a particular string to a string that is stored in a StringBuffer object.
int capacity()	Returns the capacity of a string buffer.
char charAt(int ind)	Returns the character available at a particular index from a string stored in a StringBuffer object.
int compareTo(StringBuffer strBuff)	Compares lexicographically two strings stored in two different StringBuffer objects.
StringBuffer delete(int IndBeg, int IndEnd)	Delete the characters available in a particular range of a string.
StringBuffer deleteCharAt(int ind)	Delete the character available at a particular position in a string.
void ensureCapacity(int minCap)	Sets the minimum capacity of a string buffer.
int indexOf(String string2)	Returns the index of the first occurrence of a particular substring in a string.
int indexOf(String string2, int startInd)	Returns the index of the first occurrence of a particular substring in a string where searching is started at a particular index.
StringBuffer insert(int ind, char c)	Inserts a particular character into a string.
StringBuffer insert(int ind, char chArr[ ])	Inserts a particular character array into a string.
StringBuffer insert(int ind, String string2)	Inserts a particular string into a string.
int lastIndexOf(String string2)	Returns the index of the first occurrence of a particular substring in a string.
int lastIndexOf(String string2, int startInd)	Returns the index of the last occurrence of a particular substring in a string where searching is started at a particular index in backward direction.
StringBuffer replace(int indBeg, int indEnd, String string2)	Replaces the characters of a string available at a particular range with the characters of a particular string.
StringBuffer reverse()	Reverses the character sequence in a string.

void setCharAt(int ind, char c)	Sets a particular character to a particular index of a string.
void setLength(int len)	Sets the length of a string.
String substring(int indBeg)	Returns a substring available at a particular index of a string.
String substring(int indBeg, int indEnd)	Returns a substring which contains the characters that are available in a range of a string.

Table 6.5: Some Important Methods of StringBuffer Class

The following program demonstrates the creation of string using StringBuffer class and use of some StringBuffer class methods.

Example: Java program to input two strings using StringBuffer class and to append the second string at the end of the first string using StringBuffer class method.

```
import java.util.*;
public class appendString
{
  public static void main(String[] args)
  {
   StringBuffer strBuffer1 = new StringBuffer("");
   StringBuffer strBuffer2 = new StringBuffer("");
   System.out.print("\nEnter the first string: ");
    Scanner sc = new Scanner(System.in);
    strBuffer1.append(sc.nextLine());
   System.out.print("\nEnter the second string: ");
    strBuffer2.append(sc.nextLine());
    System.out.println("\nThe first string is:");
    System.out.println(strBuffer1);
   System.out.println("\nThe second string is:");
    System.out.println(strBuffer2);
    strBuffer1.append(strBuffer2);
    System.out.println("\nAfter attachment of
    the second string,the first string is:\n");
    System.out.println(strBuffer1);
```

```
        }
    }
```

Output of the program:

```
Enter the first string: Gauhati

Enter the second string: University

The first string is:
Gauhati

The second string is:
University

After attachment of the second string,the first string is:

GauhatiUniversity
```

In the above program, 'append(String string2)' is used to append a particular string at the end of a string that is stored in a StringBuffer object.

6.9 VECTORS

In Java, vectors are the generic dynamic arrays. These arrays can store objects of any types. As vectors are dynamic arrays, these can contain any number of objects. So, the number of objects stored in a vector can be increased or decreased as required by a particular program.

In Java, Vector class is provided in java.util package to implement vectors. It implements the List interface. Vector class provides two data members to efficiently handle storage management. These members are capacity and capacityIncrement. The value of capacity is no less than the vector size. When the size of a vector becomes larger than its capacity then the capacity is automatically increased by the size which is stored in capacityIncrement. capacityIncrement is a protected data member of Vector class. Two more protected members are defined in this class that are elementCount and elementData. The value in elementCount refers the number of elements that are stored in a Vector class object. On the other hand, elementData is an array that holds the components of a vector.

The constructors of Vector class are presented in the following table.

Constructor	Job of the Constructor
Vector()	This constructor creates an empty vector. The initial capacity of this vector is 10 and standard capacity increment is 0.
Vector(int initCap)	This constructor creates an empty vector but in this case the initial capacity of the vector is passed as argument and its capacity increment is zero.
Vector(int initCap, int capInc)	This constructor also creates an empty vector but in this case, both the initial capacity and capacity increment are passed as arguments.
Vector(Collection col)	This constructor constructs a vector which stores the elements of a particular collection.

Table 6.6: Constructors of Vector Class

Some important methods of Vector class are presented in the following table.

Function Prototype	Job of the function
boolean add(S elem)	It appends a particular element to the end of a vector.
void add(int ind, S elem)	It inserts a particular element at a particular index in a vector.
int capacity()	It returns the present capacity of a vector.
void clear()	It deleted all the elements of a vector.
Object clone()	Clone of a vector is returned.
boolean contains(Object elem)	If a vector contains a particular element then it returns true.
boolean containsAll(Collection coll)	If a vector contains all the elements that are available in a particular Collection then it returns true.
void copyInto(Object[] arr)	Components of a vector are copied to a particular array.
void ensureCapacity(int capMin)	It ensures the minimum capacity of a vector and so, if requires it increases the capacity to this minimum value.

boolean equals(Object elem)	Equality of a particular object is checked with a vector.
int hashCode()	It returns the hash code of a vector.
int indexOf(Object elem)	The index of the first occurrence of a particular element in a vector is returned. If the element is not available in the vector then it returns -1.
int indexOf(Object elem, int ind)	The index of the first occurrence of a particular element in a vector is returned and in this case forward searching is performed started with a particular index. If the element is not available in the vector then it returns -1.
void insertElementAt(S ob, int ind)	It adds a particular object to a vector at a particular index.
boolean isEmpty()	It returns true if a vector is empty.
int lastIndexOf(Object elem)	The index of the last occurrence of a particular element in a vector is returned. If the element is not available in the vector then it returns -1.
int lastIndexOf(Object elem, int ind)	The index of the last occurrence of a particular element in a vector is returned and in this case backward searching is performed started with a particular index. If the element is not available in the vector then it returns -1.
boolean remove(Object elem)	It deletes the first occurrence of a particular element from a vector.
void removeAllElements()	It deletes all components of a vector and sets its size to zero.
boolean removeElement(Object elem)	It deletes the first occurrence of a particular element from a vector.
void removeElementAt(int ind)	It removes the component that is available at a particular index in a vector.
protected void removeRange(int startInd, int endInd)	It deletes all the elements that are available in a range of a vector.
void setSize(int value)	It sets the size of a vector to a value that is passed as an argument.
int size()	It gives the total number of components that are available in a vector.
Object[ ] toArray()	It returns an array that contains all the elements of a vector.
String toString()	It gives a string representation of a vector.
void trimToSize()	It reduces the capacity of a vector up to its present size.

Table 6.7: Some Important Methods of Vector Class

The following program demonstrates the creation of vector and use of some Vector class methods.

Example: Java program to create an empty vector and append new elements into it. At the end, the program will display the current capacity and size of the vector.

```
import java.util.*;
public class appendElementVector
{
    public static void main(String arg[])
    {
        //Creation of an empty Vector
        Vector vcr = new Vector();
        //Append elements into the vector
        vcr.add('A');
        vcr.add(5);
        vcr.add("KKHSOU");
        vcr.add(23.5);
       //Display elements available in the vector
        System.out.println("\nElements available
        in the vector are:\n");
        System.out.println(vcr);
        System.out.println("\nPresent size of
        the vector is :"+vcr.size());
        System.out.println("\nPresent capacity
        of the vector is :"+vcr.capacity());
     }
    }
```

Output of the program:

```
Elements available in the vector are:
[A, 5, KKHSOU, 23.5]
Present size of the vector is :4
Present capacity of the vector is :10
```

In the above program, 'add(S elem)' is used to append a particular element to the end of a vector. Then 'size()' is used to find out the size of the Vector object and 'capacity()' is used to find out the current capacity of the vector.



CHECK YOUR PROGRESS

Q 1: Which of the following class in Java can be used to create strings with flexible lengths?

(a) String
(b) StringBuffer
(c) Arrays
(d) None of the above

Q 2: Which of the following constructor creates an empty string buffer with an initial capacity of 16 characters?

(a) StringBuffer( )
(b) StringBuffer(int strLen)
(c) StringBuffer(String initString)
(d) None of the above

Q 3: Vector (int initCap) creates an empty vector with initial capacity of _____.

(a) 10
(b) 16
(c) initCap
(d) None of the above

Q 4: _____ returns capacity of a vector.

(a) ensureCapacity(int capMin)
(b) capacity()
(c) size()
(d) None of the above

Q 5: _____reduces the capacity of a vector up to its present size.

(a) trimSize()
(b) reduceSize()

(c)	trimToSize()
(d)	None of the above
Q 6:	State True/False:
(a)	capacity() returns the minimum capacity of a string buffer.
(b)	Strings created by using StringBuffer class cannot be modified.
(c)	In Java, vectors are the generic dynamic arrays.
(d)	Vector class is available in java.util package



6.10 LET US SUM UP

- Array is a data structure that store similar type of data. These data are stored in contiguous memory locations. In Java programming, arrays are container objects that store data of same types. Each element of an array is accessed by an integer value termed as subscript.
- The syntax of one-dimensional array declaration in Java is presented as follows.

  Data_Type [ ] Array_Name ;

  Or

  Data_Type Array_Name [ ] ;
- The syntax of using 'new' operator to create a one dimensional array in memory is presented as follows.

  Array_Name = new Data_Type [Array_Size] ;
- In Java, the syntax to create and initialize a one dimensional array at the time of its declaration is presented as follows.

  Data_Type [ ] Array_Name = { Data1, Data2 , Data3 , Data4,…….};
- Array with multiple subscripts is termed as multi-dimensional array.
- Two-dimensional array is an example of multi-dimensional arrays where two subscripts are required to access an array element.
- The syntax to declare a two-dimensional array in Java is presented as follows.

  Data_Type [ ][ ] Array_Name ;

  Or

  Data_Type Array_Name [ ][ ] ;

- The syntax in Java to create a two dimensional array in memory using 'new' operator is presented as follows.
  Array_Name = new Data_Type [Array_Size1][Array_Size2] ;
- 'Arrays' is a class available in the Java Collections framework under java. util package. This class contains different static methods to manipulate arrays in Java.
- String is a collection of character values. In Java, String and StringBuffer are the two classes used to handle strings efficiently.
- A string array is used to store multiple strings.
- StringBuffer class is provided to create strings with flexible lengths and the content of these strings can be modified.
- In Java, vectors are the generic dynamic arrays. Vector class is provided in java.util package to implement vectors.



6.11 FURTHER READING

1) Balaguruswamy, E. (2014). Programming with Java-A Primer. McGraw-Hill Professionals.
2) Wu, C. T. (2006). An Introduction to object-oriented programming with Java TM. Mcgraw-Hill Incorporated.
3) Schildt, H., & Coward, D. (2014). Java: the complete reference (p. 1312). New York: McGraw-Hill Education.
4) Horstmann, C. S., & Cornell, G. (2001). Core Java 2: Fundamentals (Vol. 1). Prentice Hall Professional.
5) https://docs.oracle.com/javase/tutorial/ Date of Access: 12-04-2022



6.12 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: (a)

Ans to Q No 2: (b)

Ans to Q No 3: (c)

Ans to Q No 4: (c)

Ans to Q No 5: (c)

Ans to Q No 6: (a) Subscript

Ans to Q No 6: (b) toString(char chArr[ ])

Ans to Q No 7: (c) a hash code

Ans to Q No 8: (d) the character sequence that is available in a particular character array

Ans to Q No 9: (e) charAt (int Ind)

Ans to Q No 10: (b)

Ans to Q No 11: (a)

Ans to Q No 12: (c)

Ans to Q No 13: (b)

Ans to Q No 14: (c)

Ans to Q No 15: (a) False

Ans to Q No 16: (b) False

Ans to Q No 17: (c) True

Ans to Q No 18: (d) True



6.13 MODEL QUESTIONS

Q 1: How a two-dimensional array can be created in memory by using Java statement?

Q 2: What is the purpose of Arrays class?

Q 3: Write a Java program to merge two integer arrays so that the resultant array becomes sorted.

Q 4: How strings can be handled in Java?

Q 5: Write down the different constructors of String class.

Q 6: Write a Java program to check an input string is palindrome or not palindrome.

Q 7: What are the differences between String class and StringBuffer class?

Q 8: What is vector? How vector can be created and manipulated in Java?

Q 9: What is the difference between an array and a vector?

*** ***** ***

UNIT 7: INTERFACE AND PACKAGES

UNIT STRUCTURE

7.1 Learning Objectives

7.2 Introduction

7.3 Introduction to Interface

- 7.3.1 Defining an Interface
- 7.3.2 Implementing an Interface
- 7.3.3 Nested Interface

7.4 Package

- 7.4.1 Defining a Package
- 7.4.2 Finding Packages and CLASSPATH
- 7.4.3 Access Protection in Packages
- 7.4.4 Importing Packages

7.5 Let Us Sum Up

7.6 Further Reading

7.7 Answer to Check Your Progress

7.8 Model Questions

7.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about the concept of multiple inheritance through Interface
- acquire the process of defining and implementing Interface
- get the usefulness of a package
- learn the process of defining and using package

7.2 INTRODUCTION

This unit describes two of Java's most advanced features: packages and interfaces. There is no concept of multiple-inheritance in Java. Instead of this, Interface concept is included in Java which is unique to the language. Another unique feature in Java is Packages. Packages are containers for classes which are meant for keeping the class name space

compartmentalized. By organizing the user's classes into packages, they can be reused to write other classes. Packages are stored in a hierarchical manner and are explicitly imported into new class definitions. For example, a package permits us to create a class named P1, which we can store in our own package.

7.3 INTRODUCTION TO INTERFACE

An interface in Java is essentially a special kind of class. Like classes, interfaces contain methods and variables. However, unlike classes, interfaces are always completely abstract. Through the use of the interface, Java allows us to fully abstract the interface from its implementation. Using interface, we can specify a set of methods that can be implemented by one or more classes. The interface, itself, does not actually define any implementation. They are similar to abstract classes, but interface has an advantage over an abstract class. A class can implement more than one interface wherein, a class can only inherit a single superclass (abstract or otherwise). Interfaces are designed to support dynamic method resolution at run time. Normally, in order for a method to be called from one class to another, both classes need to be present at compile time. Interfaces are designed to disconnect the definition of a method or set of methods from the inheritance hierarchy.

7.3.1 Defining an Interface

Similar to a class declaration, an interface is defined. The general form of an interface is:

```
 interface interface_name
{
type final-varname1 = value;
type final-varname2 = value;
//…
type final-varnameN = value;
return-type method-name1(parameter-list);
```

return-type method-name2(parameter-list);

// ...

return-type method-nameN(parameter-list);

}

Although member elements are present in an interface, they are static and final, so they cannot be changed by the implementing class. They must be initialized. Again, methods are only declared in an interface; the code to implement the method is not written. They end with a semicolon after the parameter list. They are, essentially, abstract methods; there can be no default implementation of any method specified within an interface. Each class that includes an interface must implement all of the methods. All methods and variables inside an interface are implicitly public. While classes can inherit from only one super class, interfaces are able to inherit from as many as interface one can provide.

After defining an interface, it is required to implement with classes which can be done with the following syntax:

```
class  ClassName [ extends SuperClassName ]
[ implements InterfaceName1 [, Interface Name2] )
 {
                class body
 }
```

7.3.2 Implementing Interfaces

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, the implements clause is used in a class definition, and then create the methods defined by the interface.

The general form of a class that includes the implements clause looks like:

class classname[extends superclass][implements interface [,interface...]]

```
{
    // class-body
}
```

If a class implements more than one interface, the interfaces are separated with a comma. If a class implements two interfaces that declare the same method, then the same method will be used by clients of either interface. The methods that implement an interface must be declared public. Also, the type signature of the implementing method must match exactly the type signature specified in the interface definition.

An example of an interface and its implementation is given below:

```
interface Resume
{
  void bioData( ) ; // By default it is an abstract
method.
            // when it is declared in an interface

}
class Teacher extends Employee implements Resume {
  // Assume Employee is a class defined somewhere
String Qualification;// A  member element

experience( ) {  …..  } // A member function
public void bioData( )
{
  // Code for this method for preparing the bio-
data
  of a teacher //
}
}
class Student implements Resume {
String Discipline;    // A member element.
```

```
result( ){ …… } // A member function
public void bioData ( ) {
// Code for the method for preparing the bio-data
   of a student //
}
}
```

Accessing Implementations through Interface References

We can declare variables as object references that use an interface rather than a class type. Any instance of any class that implements the declared interface can be referred to by such a variable. When a method is called by using one of these references, the correct one will be called based on the actual instance of the interface being referred to. This is one of the key features of interfaces. The method to be executed is chosen dynamically at run time, allowing classes to be created rather than the code which calls methods on them. The calling code can dispatch through an interface without the knowledge of anything about the "callee."

Here is an example which calls a method named callback( ) via an interface reference variable:

```
class Test {
public static void main(String args[])
{
   Callback c = new Client();
   c.callback(60);
}
}
```

The output of this program will be:

callback called with 60

Notice that variable c is declared to be of the interface type Callback, yet it was assigned an instance of Client. Although c can be used to access the callback( ) method, it cannot access any other members of the Client class. The object reference c cannot be used to access other methods defined by Client.

Partial Implementations:

If a class includes an interface but does not fully implement the methods defined by that interface, then that class must be declared as abstract.

For example:

```
abstract class Test1 implements Callback
{
  int x, y;
  void show()
  {
   System.out.println(x + " " + y);
  }
}
```

This example shows that the class Test1 does not implement callback( ) and must be declared as abstract. Any class that inherits Test1 must implement callback( ) or be declared abstract itself.

7.3.3 Nested Interfaces

An interface which can be declared a member of a class or another interface is called a member interface or a nested interface. A nested interface can be declared either as public, private, or protected. When a nested interface is used outside of its scope, it must be qualified by the name of the class or interface of which it is a member. Thus, outside of the class or interface in which a nested interface is declared, its name must be fully qualified.

The following example demonstrates the concept of a nested interface:

```
// A nested interface example.
// This class contains a member interface.
class A {
public interface NestedIF
{
```

```
  boolean negative(int a);
}
}
// B implements the nested interface.
class B implements A.NestedIF
{
    public boolean negative(int a)
    {
        return a < 0 ? false : true;
    }
}
class Main
{
   public static void main(String args[])
  {
   // use a nested interface reference
   A.NestedIF nif = new B();
   if(nif.negative(10))
   System.out.println("10 is not negative");
   if(nif.negative(-12))
   System.out.println("Yes, it is negative");
  }
}
```

Notice that A defines a member interface called NestedIF and that it is declared public. Next, B implements the nested interface by specifying

implements A.NestedIF

Moreover, the name is fully qualified by enclosing class name. Inside the main( ) method, an A.NestedIF reference named nif is created, and it is assigned a reference to a B object. Because B implements A.NestedIF, which is legal.

Applying Interfaces:

To understand the power of interfaces, let's consider another

practical example. We can consider a class called Stack that implements the functions of a simple fixed-size stack. No matter how the stack is implemented, the interface to the stack will remain the same where the methods push( ) and pop( ) define the interface to the stack independently of the details of the implementation.

```
// Define an integer stack interface.
interface IntStack
{
void push(int item); // store an item
int pop(); // retrieve an item
}
```

The following program creates a class called Stack that implements a fixed-length version of an integer stack:

```
// An implementation of IntStack that uses fixed storage.
class Stack implements IntStack
{
private int stck[];
private int tos;
// allocate and initialize stack
Stack(int size)
{
stck = new int[size];
tos = -1;
}
// Push an item onto the stack
public void push(int item)
{
if(tos==stck.length-1) // use length member
System.out.println("Stack is full.");
else
```

```
stck[++tos] = item;
}
// Pop an item from the stack
public int pop()
{
   if(tos < 0) {
    System.out.println("Stack underflow.");
   return 0;
}
else
   return stck[tos--];
}
}
class IFTest
{
   public static void main(String args[])
  {
    Stack mys1 = new Stack(10);
    Stack mys2 = new Stack(15);
   // push some numbers onto the stack
   for(int i=0; i<10; i++)
   mys1.push(i);
   for(int i=0; i<15; i++)
   mys2.push(i);
// pop those numbers off the stack
   System.out.println("Stack in mystack1:");
   for(int i=0; i<10; i++)
   System.out.println(mys1.pop());
   System.out.println("Stack in mystack2:");
   for(int i=0; i<15; i++)
   System.out.println(mys2.pop());
  }
}
```

Interfaces Can Be Extended:

One interface can inherit another one by using the keyword extends. The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain. Following example explains it:

```
// One interface can extend another.
interface A
{
  void meth1();
  void meth2();
}
// B now includes meth1() and meth2()
interface B extends A
{
  void meth3();
}
// This class must implement all of A and B
class MyClass implements B
{
   public void meth1()
    {
     System.out.println("Implementing meth1().");
    }
    public void meth2()
    {
    System.out.println("Implementing meth2().");
 }
 public void meth3()
 {
    System.out.println("Implementing meth3().");
 }
}
```

```
class Interface_Extend
{
   public static void main(String arg[])
   {
      MyClass ob = new MyClass();
      ob.meth1();
      ob.meth2();
      ob.meth3();
  }
}

/* Example on the multiple inheritance with
interface with implementing two or mor interfaces*/
class A {
   protected int i = 100;
   public void print() { System.out.println("I am
from A "+i);
    } }
interface C {
    public static int j = 500;
    void printInterfaceC();
}
interface D {
    public static int k = 600;
    void printInterfaceD();
}
class B extends A implements C, D{
     public int aValue = 900;
     public void printInterfaceC() {
          System.out.println("I am from C " + j);
     }
     public void printInterfaceD() {
          System.out.println("I am from D " + k);
```

```
        }
        public void printB() {
            super.print();
            printInterfaceC();
            printInterfaceD();
        }
}
class Demonstration {
   public static void main (String[] args){
            B b = new B();
            b.printB();
   } }
```



CHECK YOUR PROGRESS

Q 1: Choose the correct options:

i) Which of these can be used to fully abstract a class from its implementation?

a) Objects b) Packages c) Interfaces

d) None of the Mentioned

ii) Which of these access specifiers can be used for an interface?

a) Public b) Protected c) private

d) All of the mentioned

iii) Which of the following is the correct way of implementing an interface A by class B?

a) class B extends A{} b) class B implements A{}

c) class B imports A{} d) None of the mentioned

iv) All methods must be implemented of an interface.

a) True b) False

7.4 PACKAGES

In the preceding examples, the name of each class was taken from the same name space. This means that a unique name has to be used for each class to avoid name collisions. Gratefully, Java includes a mechanism for partitioning the class name space into more manageable chunks. This is known as the package. The package is both a naming and a visibility control mechanism. We can define many classes inside a package that are not accessible by code outside that package. Moreover, we can declare class members that are only accessible to other members of the same package. Thus, it allows us classes to have intimate knowledge of each other, but not to expose that knowledge to the rest of the world.

There are two types of packages available in Java which are:

Built-in packages

These packages are already defined in java API, for example, java.util, java.io,java,lang, java.awt, java.applet, java.net, etc.

User defined packages

When user creates a package according to his/her need, then it is called user defined package.

7.4.1 Defining a Package

To create a package, we have to include a **package** command as the first statement in a Java source file. Any classes declared within that file will belong to the specified **package**. Actually, the package statement defines a name space in which classes are stored. In absence of package statement, the class names are put into the default **package**, which has no name.

This is the general form of the **package** statement:

package pkg_name;

Here, pkg_name is the name of the package.

For example, the following statement creates a package **MyPackage.**

package MyPackage;

Java uses file system directories to store packages. For example, the .class files for any classes declared to be part of MyPackage and must be stored in a directory called MyPackage. It is to be noted that case is significant, and the directory name must match the package name exactly. More than one file can include the same package statement. The package statement simply depicts to which package the classes defined in a file belong. It does not exclude other classes in other files from being part of that same package. Most real-world packages are spread across many files.

Java allows us to create a hierarchy of packages. To order to do so, simply separate each package name
from the one above it by use of a period. The general form of a multileveled package statement is shown here:

package pkg1[.pkg2[.pkg3]];

A package hierarchy must be reflected in the file system in Java development system. For example, a package declared as

package java.awt.image;

needs to be stored in java\awt\image in a Windows environment. It is to be noted that we cannot rename a package without renaming the directory in which the classes are stored.

7.4.2 Finding Packages and CLASSPATH

As explained above, packages are mirrored by directories. Now one important question is: How does the Java run-time system know where to look for packages that you create? For this query, there are three answers. First, by default, the Java run-time system uses the current working directory as its starting point.

Secondly, we can specify a directory path or paths by setting the CLASSPATH environmental variable. Lastly, Java allows us to use the - classpath option with Java and javac to specify the path to the classes.

For example, consider the following package specification:

package MyPackage

In order for a program to find MyPackage, one of three things must be true. Either the program can be executed from a directory immediately above MyPackage, or the CLASSPATH must be set to include the path to MyPackage, or the -classpath option must specify the path to MyPackage when the program is run via java.

For other two options, the class path must not include MyPackage itself. It is to specify simply the path to MyPackage. For example, in a Windows environment, if the path to MyPackage is

C:\MyPrograms\Java\MyPackage

Then the class path to MyPackage is

C:\MyPrograms\Java

This is the approach used in the following example.

A Short Package Example

```
// A simple package
package MyPackage;
class BalanceAmount
{
   String name;
   double bal;
   BalanceAmount(String n, double b)
   {
     name = n;
     bal = b;
   }
   void show()
   {
      if(bal<0)
      System.out.print("--> ");
      System.out.println(name + ": $" + bal);
   }
}
class Account_Balance
{
```

```
public static void main(String args[])
{
   BalanceAmount savings[]= new Balance[3];
  savings[0] = new BalanceAmount("Ramesh Murthi",
5460.67);
    savings[1] = new BalanceAmount("Williamson",
3257.02);
    savings[2] = new BalanceAmount("Amit Kohli",
456.78);
  for(int i=0; i<3; i++) savings[i].show();
 }
}
```

Now, we have to call this file Account_Balance.java and put it in a directory called MyPackage. Next, we have to compile the file. Make sure that the resulting .class file is also in the MyPackage directory. Then, try to execute the Account_Balance class, using the following command line:

java MyPackage.Account_Balance

7.4.3 Access Protection in Packages

It is clear from the above discussion that access to a private member of a class is granted only to other members of that class. Packages provide another dimension to access control. Java provides many levels of protection to allow fine-grained control over the visibility of variables and methods within classes, subclasses, and packages. Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods. Packages act as containers for classes and other subordinate packages whereas Classes act as containers for data and code. Thus, the class is considered Java's smallest unit of abstraction. Because of the interplay between classes and packages, Java provides 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 namely private, public, and protected, provide various ways of access protection. Table 1 presents the summary of the class members accessibility. While Java's access control mechanism may seem complicated, we can simplify it as follows. Anything declared public can be accessed from anywhere. Anything declared private cannot be seen outside of its class. When a member does not have an explicit access specification, it is visible to subclasses as well as to other classes in the same package. This is the default access. If we want to allow an element to be seen outside the current package, but only to classes that subclass our class directly, then declare that element protected.

A non-nested class has only two possible access levels: default and public. When a class is declared as public, it is accessible by any other code. If a class has default access, then it can only be accessed by other code within its same package. When a class is public, it must be the only public class declared in the file, and the file must have the same name as the class.

Table 1. Accessibility of Class members

	Private	**Public**	**Protected**	**No modifier**
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	Yes	Yes	No
Different Package non-subclass	No	Yes	No	No

An Example of Access control

The following example shows all combinations of the access control modifiers. This example has two packages and five classes. Remember that the classes for the two different packages require to be stored in directories named after their respective packages-say, p1 and p2. The source for the first package p1 defines three classes: Protection, Derived, and SamePackage. The Protection class defines four int variables in each of the legal protection modes. The variable n is declared with the default protection, n_priv is private, n_prot is protected, and n_publ is public. Each subsequent class tries to access the variables as an instance of this class. The lines that cannot be compiled due to access restrictions are put as comments.

The second class, Derived, is a subclass of Protection in the same package, p1. This permits Derived class to access to every variable in Protection except for n_priv, the private one.

The third class, SamePackage, is not a subclass of Protection, but is in the same package and also has access to all except n_priv.

This is the file **Protection.java:**

```
package p1;
public class Protection
{
int n = 1;
private int n_priv = 2;
protected int n_prot = 3;
public int n_publ = 4;
public Protection() {
System.out.println("base constructor");
System.out.println("n = " + n);
System.out.println("n_pri = " + n_priv);
System.out.println("n_pro = " + n_prot);
```

```
System.out.println("n_pub = " + n_publ);
}
}
```

This is file **Derived.java:**

```
package p1;
class Derived extends Protection
{
Derived() {
System.out.println("derived constructor");
System.out.println("n = " + n);
// class only
// System.out.println("n_pri = "4 + n_priv);
System.out.println("n_prot = " + n_prot);
System.out.println("n_publ = " + n_publ);
}
}
```

This is file **SamePackage.java:**

```
package p1;
class SamePackage
{
SamePackage() {
Protection p = new Protection();
System.out.println("same package
constructor");
System.out.println("n = " + p.n);
// class only
// System.out.println("n_priv = " + p.n_priv);
System.out.println("n_prot = " + p.n_prot);
System.out.println("n_publ = " + p.n_publ);
}    }
```

Following is the source code for the second package, p2. The two classes defined in p2 cover the other two conditions that are affected by access control. The first class, Protection2, is a

subclass of p1.Protection. This allows to access to all of p1.Protection's variables except for n_priv and n, the variable declared with the default protection. It is to be mentioned that the default only allows access from within the class or the package, not extra-package subclasses. Finally, the class OtherPackage has access to only one variable, n_publ, which was declared public.

This is file **Protection2.java:**

```
package p2;
class Protection2 extends p1.Protection
{
Protection2() {
System.out.println("derived other
package constructor");
// class or package only
// System.out.println("n = " + n);
// class only
// System.out.println("n_pri = " + n_priv);
System.out.println("n_prot = " + n_prot);
System.out.println("n_publ = " + n_publ);
}
}
```

This is file **OtherPackage.java:**

```
package p2;
class OtherPackage
{
OtherPackage() {
p1.Protection p = new p1.Protection();
System.out.println("other package constructor");
// class or package only
// System.out.println("n = " + p.n);
// class only
// System.out.println("n_priv = " + p.n_priv);
// class, subclass or package only
```

```
        // System.out.println("n_prot = " + p.n_prot);
        System.out.println("n_publ = " + p.n_publ);
        }
        }
```

If we want to use these two packages, here are two test files we can use. First one for p1 is:

```
        // Demo package p1.
        package p1;
        // Instantiate the various classes in p1.
        public class Demo
        {
        public static void main(String args[])
        {
        Protection ob1 = new Protection();
        Derived ob2 = new Derived();
        SamePackage ob3 = new SamePackage();
        }
        }
```

The test file for p2 is shown next:

```
        // Demo package p2.
        package p2;
        // Instantiate the various classes in p2.
        public class Demo {
        public static void main(String args[]) {
        Protection2 ob1 = new Protection2();
        OtherPackage ob2 = new OtherPackage();
        }
        }
```

7.4.4 Importing Packages

There are no core Java classes in the unnamed default package. All of the standard classes are stored in some named package.

Since classes within packages must be fully qualified with their package name or names, it could become tedious to type in the long dot-separated package path name for every class you want to use. Due to this, Java includes the import statement to bring certain classes, or entire packages, into visibility. Once imported, a class can be referred to directly, using only its name.

In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. The general form of the import statement is:

```
import pkg1[.pkg2].(classname|*);
```

Here, pkg1 is the name of a top-level package, and pkg2 is the name of a subordinate package inside the outer package separated by a dot (.). There is no limit on the depth of a package hierarchy, except that imposed by the file system. Finally, we specify either an explicit classname or a star (*), which indicates that the Java compiler should import the entire package. This code fragment shows both forms in use:

```
import java.util.Date;
import java.io.*;
```

All of the standard Java classes included with Java are stored in a package called java. The basic language functions are stored in a package inside of the java package called java.lang. Normally, we have to import every package or class that we want to use, but since Java is useless without much of the functionality in java.lang, it is implicitly imported by the compiler for all programs. This is equivalent to the following line:

```
import java.lang.*;
```

If a class with the same name exists in two different packages that we import using the * form, the compiler will remain silent, unless we try to use one of the classes. In that case, we will get a compile-time error and have to explicitly name the class specifying its package. It must be emphasized that the import statement is optional. Any place we use a class name, we can use its fully qualified

name, which includes its full package hierarchy. For example, the following program segment uses an import statement:

```
import java.util.*;
class MyDate extends Date { }
```

The same example without the import statement looks like this:

```
class MyDate extends java.util.Date
{
}
```

In this version, Date is fully-qualified.

When a package is imported, only those items within the package declared as public will be available to non-subclasses in the importing code. For example,

if we want the BalanceAmount class of the package MyPackage shown earlier to be available as a stand-alone class for general use outside of MyPackage, then it is required to declare it as public and put it into its own file, as shown here:

```
package MyPackage;
public class BalanceAmount {
String name;
double bal;
public BalanceAmount (String n, double b) {
name = n;
bal = b;
}
public void show() {
if(bal<0)
System.out.print("--> ");
System.out.println(name + ": $" + bal);
}
}
```

Here, the BalanceAmount class is now public. Also, its constructor and its show( ) method are also public. This means that

they can be accessed by any type of code outside the MyPackage package. For example, here TestBalance imports MyPackage and is then able to make use of the BalanceAmount class:

```
import MyPackage.*;
class TestBalance {
public static void main(String args[]) {
/* Because Balance is public, you may use Balance
class and call its constructor. */
BalanceAmount test = new BalanceAmount("J. K.
Rashmi", 84.78);
test.show(); // you may also call show()
}
}
```



CHECK YOUR PROGRESS

Q 2: Select the correct option for the following questions:

i) Which of these keywords is used to define packages in Java?

a) pkg b) Pkg c) package

d) Package

ii) Which of these is a mechanism for naming and visibility control of a class and its content?

a) Object b) Packages c) Interfaces

d) None of the Mentioned

iii) Which of these access specifiers can be used for a class so that its members can be accessed by a different class in the same package?

a) Public b) Protected c) No Modifier

d) All of the mentioned

iv) What is the maximum number of Java classes that can be kept inside a Java package?

a) 8 b) 64 c) 128 d) unlimited



7.5 LET US SUM UP

- An interface in Java is essentially a special kind of class. Like classes, interfaces contain methods and variables. However, unlike classes, interfaces are always completely abstract.
- To define an interface, Java provides a keyword interface which is followed by interface name. Within an interface, variables and methods can be initialized. However, no method can be defined inside a package.
- All the variables inside an interface are declared as final and static.
- Once an interface has been defined, one or more classes can implement that interface. To implement an interface, the implements clause is used in a class definition, and then create the methods defined by the interface.
- Java includes a mechanism for partitioning the class name space into more manageable chunks. This is known as the package. The package is both a naming and a visibility control mechanism.
- A Package can group the similar type of classes, interfaces and sub-packages and provide access control. It organizes classes into single unit.
- There are two types of packages: Built-in and User defined.
- Packages provide another dimension to access control. Java provides many levels of protection to allow fine-grained control over the visibility of variables and methods within classes, subclasses, and packages. Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods. Packages act as containers for classes and other subordinate packages whereas Classes act as containers for data and code.



7.6 FURTHER READINGS

1) Schildt, H., & Coward, D. (2014). Java: the complete reference. New York: McGraw-Hill Education.
2) Rao, N., Core and Advanced Java Black Book. dreamTech

7.7 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: i) c ii) d iii) b iv) a

Ans to Q No 2: i) c ii) b iii) b iv) d

7.8 MODEL QUESTIONS

Q 1: What is an interface? How to define it? Discuss with an example.

Q 2: How can we access an interface from another interface and how to access from a class?

Q 3: Explain extending interfaces with a proper example ?

Q 4: What is a Package? Explain the Packages with a suitable example and how to import packages?

Q 5: Describe the levels of access protection available for packages.

*** ***** ***

UNIT 8: THREADS

UNIT STRUCTURE

8.1 Learning Objectives

8.2 Introduction

8.3 Introduction to New Thread

8.4 Creating New Threads by Extending the Thread Class

8.5 Creating a Thread by Implementing Runnable Interface Static Members

8.6 Threads in the Running State

8.7 Sleeping and Interruptions

8.8 Signaling with Wait and Notify

8.9 Let Us Sum Up

8.10 Further Reading

8.11 Answer to Check Your Progress

8.12 Model Questions

8.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about thread in Java
- describe process of creation of thread
- learn about thread State

8.2 INTRODUCTION

In the previous unit, we have discussed the concept of packages and interface in Java programming language. This unit will introduce you to the concept of threads. We will discuss how to create a new thread at the beginning and various concept associated with threads.

8.3 INTRODUCTION TO NEW THREAD

Thread is the smallest independent unit of a process, it is a lightweight subprocess. While executing a Java program, it follows a path

and that path or the direction followed by a Java program during execution is called a thread. Every Java program has at least one thread.

A thread when created, by default, is a new thread. The lifecycle of a thread starts with a new thread state. The codes are not executed or initiated in this state.

A thread in Java can be created in the following two ways:

- Extending java.lang.Thread class
- Implementing Runnable interface

The number of services assigned to a given thread is referred to as its priority. Any thread generated in the JVM is given a priority. The priority scale runs from 1 to 10.

- 1 is known as the lowest priority.
- 5 is known as standard priority.
- represents the highest level of priority.

8.4 CREATING NEW THREADS BY EXTENDING THE THREAD CLASS

Threads can be created by extending the thread class. The thread class is extended by creating a new class.

Program 8.1:

```
import java.io.*;
import java.util.*;
public class ABC extends Thread
{
      // initiated run method for Thread
public void run()
{
System.out.println("Thread Started Running...");
}
public static void main(String[] args)
{
      ABC g1 = new ABC();
```

```
                // invoking Thread
                g1.run();
        }
}
```



CHECK YOUR PROGRESS

Q 1: What is valid constructors for Thread?

.....

Q 2: What is the name of the method used to start a thread execution?

.....

Q 3: What is run() method in thread?

.....

8.5 CREATING A THREAD BY IMPLEMENTING RUNNABLE INTERFACE STATIC MEMBERS

Threads can also be created by implementing interfaces.

Program 8.2:

```
import java.io.*;
import java.util.*;
public class ABC implements Runnable
 {
      // method to start Thread
      public void run()
      {
            System.out.println(
                  "Thread is Running Successfully");
      }
      public static void main(String[] args)
      {
            ABC g1 = new ABC();
            // initializing Thread Object
```

```
                Thread t1
                      = new Thread(g1);
                t1.run();
            }
        }
```

8.6 THREADS IN THE RUNNING STATE

When the processor assigns a time slot to the thread, it enters the running state. A thread from the runnable state is chosen for execution by the thread scheduler and the process to run the thread starts in this state. Only a thread from a runnable state could enter the running state. The CPU allots the time for execution of the thread in the running state.

The runnable state of a thread is a state in which the thread is ready to run is said to be in a Runnable state It is the responsibility of the thread scheduler to give the thread, time to run.

It is the second phase of a new-born thread is the execution phase. When the start() method is called on a the new instance of a thread, it enters into a runnable state. In the runnable state, thread is ready for execution and is waiting for availability of the processor (CPU time). There are many threads that are ready for execution, they all are waiting in a queue (line). If all threads have equal priority, a time slot is assigned for each thread execution on the basis of first-come, first-serve manner by CPU. The process of allocating time to threads is known as time slicing. A thread can come into runnable state from running, waiting, or new states.

8.7 SLEEPING AND INTERUPT

Sleep is an important method to help the other threads to get executed, that is to make processing time available for other threads in the process. The sleep() method makes the current thread stop and suspend its execution for a specific period of time.

Program 8.3:

```
// important import statements
```

```
import java.lang.Thread;
import java.io.*;
public class TestSleepMethod2
{
   // main method
   public static void main(String argvs[])
   {
    try
    {
      for (int j = 0; j < 5; j++)
      {
       // The main thread sleeps for the 100
       //milliseconds, which is 1 sec
       // whenever the loop runs
        Thread.sleep(100);
       // displaying the value of the variable
       System.out.println(j);
     }
   }
   catch (Exception expn)
   {
    // catching the exception
    System.out.println(expn);
   }
  }
}
```

Interrupt is also an important method in thread execution, as it helps to interrupt the thread in execution. If a thread is sleeping or waiting, the interrupt() method helps to call back the thread.

Program 8.4:

```
class TestInterruptingThread2 extends Thread
{
        public void run()
```

```
{
Try
{
   Thread.sleep(100);
   System.out.println("task");
}
catch(InterruptedException e)
{
    throw new RuntimeException("Thread
     interrupted..."+e);
}

}
public static void main(String args[])
{
    TestInterruptingThread2 t2=new
    TestInterruptingThread2();
    t2.start();
    Try
    {
        t1.interrupt();
    }
    catch(Exception e)
    {
      System.out.println("Exception handled "+e);
   }
  }
}
```

8.8 SIGNALING WITH WAIT AND NOTIFY

In Java programming, when the wait() method is called on an object, the thread becomes inactive.it forces the current thread to wait, until it is

called.

notify() is the method which when called on an object activates the wait() in the thread. notify() wakes up the waiting threads.

Two threads having access to the same instance of a class can be used to pass signals. Here is an example where, one thread will call the doWait() method, and the other thread will call doNotify() method :

```
public class MachineObject
{
}
public class MyWaitNotify
{
MonitorObject myMachineObject = new MachineObject();
public void doWait()
{
    synchronized(myMachineObject)
{
      Try
{
        myMachineObject.wait();
}
catch(InterruptedException e){...}
}
}
public void doNotify()
{
    synchronized(myMachineObject)
{
      myMachineObject.notify();
   }
  }
 }
```



CHECK YOUR PROGRESS

Q 5: When the event for which a thread is blocked occurs?

.....

Q 6: What is the resume()?

.....

Q 7: Thread synchronization is required because all threads of a process share the same address space (True or False)

.....

Q 8: What is wait () method in thread?

.....

Q 9: What is notify() method?

.....



8.9 LET US SUM UP

- Thread is the smallest independent unit of a process, it is a lightweight subprocess. While executing a Java program, it follows a path and that path or the direction followed by a Java program during execution is called a thread. Every Java program has at least one thread.
- A thread in Java can be created in the following two ways:
  - Extending java.lang.Thread class
  - Implementing Runnable interface
- Threads can be created by extending the thread class. The thread class is extended by creating a new class.
- Threads can also be created by implementing interfaces.
- When the processor assigns a time slot to the thread, it enters the running state. A thread from the runnable state is chosen for execution by the thread scheduler and the process to run the thread starts in this state. Only a thread from a runnable state could enter the running state. The CPU allots the time for execution of the thread in the running state.
- The runnable state of a thread is a state in which the thread is ready to

run is said to be in a Runnable state It is the responsibility of the thread scheduler to give the thread, time to run.

- In java programming, when the wait() method is called on an object, the thread becomes inactive.it forces the current thread to wait, until it is called.
- notify() is the method which when called on an object activates the wait() in the thread. notify() wakes up the waiting threads.
- Sleep is an important method to help the other threads to get executed, that is to make processing time available for other threads in the process. The sleep() method makes the current thread stop and suspend its execution for a specific period of time.
- Interrupt is also an important method in thread execution, as it helps to interrupt the thread in execution. If a thread is sleeping or waiting, the interrupt() method helps to call back the thread.

8.10 FURTHER READINGS

1) Schildt, H. (2020). The complete reference Java.

8.11 ANSWER TO CHECK YOUR PROGRESS

Ans to Q No 1: Thread()

Ans to Q No 2: Start()

Ans to Q No 3: The run() method of a thread is like the main() method to an application. Starting the thread causes the object's run method to be called in that separately executing thread.

Ans to Q No 4: When thread moves to the ready queue.

Ans to Q No 5: The resume() method is deprecated. It resumes a suspended thread.

Ans to Q No 6: True

Ans to Q No 7: In java programming, when the wait() method is called on an object, the thread becomes inactive.it forces the current thread to wait, until it is called.

Ans to Q No 8: notify() is the method which when called on an object activates the wait() in the thread. notify() wakes up the waiting threads.



8.12 MODEL QUESTIONS

Q 1: What is thread in Java?

Q 2: How thread is intiate in Java?

Q 3: Explain the wait() and notify() in Java thread.

Q 4: Explain about the Java Thread Priorities.

*** ***** ***

UNIT 9: THE JAVA.LANG.MATH CLASS

UNIT STRUCTURE

9.1 Learning Objectives

9.2 Introduction

9.3 About java.lang.Math class

9.3.1 Methods of the java.lang.Math Class

9.3.2 Program Examples

9.4 Let Us Sum Up

9.5 Further Reading

9.6 Answer to Check Your Progress

9.7 Model Questions

9.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe the necessity of java.lang.Math class
- use various methods under java.lang.Math class
- use various logarithmic methods
- use various angular and trigonometric methods

9.2 INTRODUCTION

Java language provides various packages and built-in classes for use as per the requirements. In the last unit of the first block, the concept of thread and creating a thread was elaborately discussed. In this unit, we will explain about java.lang.Math class and various methods under this class. These methods are used for various mathematical operations. The next unit explores the concept of exception handling.

9.3 ABOUT JAVA.LANG.MATH CLASS

The java.lang package defines classes that are fundamental to the Java language. For this reason, all classes are in java.lang packages are imported automatically, so there is no reason to write an import statement

for them. The java.lang package also contains the Math class, which is used to perform basic mathematical operations such as the elementary exponential, logarithm, square root, and trigonometric functions. The Math class defines two mathematical constants PI and E. Their meaning is as follows:

- **static double E** - This is the double value that is closer than any other to e, the base of the natural logarithms.
- **static double PI** - This is the double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter.

Declaration of these two constants is:

```
public final static double Math.PI
public final static double Math.E
```

Because all methods of the Math class are defined as static, it doesn't need to create an instance to use them. It's not possible to create an instance of the Math class because the constructor is private. Further, we can't extend the Math class either, because it's marked final.

9.3.1 Methods of the java.lang.Math Class

Java.lang.Math class provides several methods to work on mathematical operations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs(),sqrt(), etc. This class inherits methods from java.lang.Object class.

The methods of the Math class are static and are accessed through the class name like any static method. For these method calls, the general form is:

```
result = Math.aStaticMathMethod();
```

The following table enlists the various Math methods and their functions.

Table 9.1: Various methods under Math class

Method Name	Description
Math.abs()	Returns the Absolute value of the given value.

Math.max()	Returns the Largest of two values.
Math.min()	Finds the Smallest of two values.
Math.round()	used to round of the decimal numbers to the nearest value.
Math.sqrt()	Returns the square root of a number.
Math.cbrt()	used to return the cube root of a number.
Math.pow()	Returns the value of first argument raised to the power to second argument.
Math.signum()	used to find the sign of a given value.
Math.ceil()	Finds the smallest integer value that is greater than or equal to the argument or mathematical integer.
Math.copySign()	used to find the Absolute value of first argument along with sign specified in second argument.
Math.nextAfter()	Returns the floating-point number adjacent to the first argument in the direction of the second argument.
Math.nextUp(dd)	It returns the floating-point value adjacent to d in the direction of positive infinity.
Math.nextDown(d)	It returns the floating-point value adjacent to d in the direction of negative infinity.
Math.floor()	used to find the largest integer value which is less than or equal to the argument and is equal to the mathematical integer of a double value.
Math.floorDiv()	finds the largest integer value that is less than or equal to the algebraic quotient.
Math.random()	returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
Math.rint()	returns the double value that is closest to the given argument and equal to mathematical integer.
Math.hypot()	Computessqrt(x^2 +y^2) without intermediate overflow or underflow.
Math.getExponent()	returns the unbiased exponent used in the representation of a value.
Math.IEEEremainder()	used to calculate the remainder operation on two arguments as prescribed by the IEEE 754 standard and returns value.

Math.addExact()	returns the sum of its arguments, throwing an exception if the result overflows an int or long.
Math.subtractExact()	returns the difference of the arguments, throwing an exception if the result overflows an int.
Math.multiplyExact()	used to return the product of the arguments, throwing an exception if the result overflows an int or long.
Math.incrementExact()	returns the argument incremented by one, throwing an exception if the result overflows an int.
Math.decrementExact()	returns the argument decremented by one, throwing an exception if the result overflows an int or long.
Math.negateExact()	returns the negation of the argument, throwing an exception if the result overflows an int or long.
Math.toIntExact()	returns the value of the long argument, throwing an exception if the value overflows an int.

Moreover, Math class includes some logarithmic, angular and trigonometric functions as listed below.

Logarithmic Math Methods

Method Name	Description
Math.log()	It returns the natural logarithm of a double value.
Math.log10()	It is used to return the base 10 logarithm of a double value.
Math.log1p()	It returns the natural logarithm of the sum of the argument and 1.
Math.exp()	It returns E raised to the power of a double value, where E is Euler's number and it is approximately equal to 2.71828.
Math.expm1()	It is used to calculate the power of E and subtract one from it.

Trigonometric Math Methods

Method Name	Description
Math.sin()	It is used to return the trigonometric Sine value of a Given double value.

Math.cos()	It is used to return the trigonometric Cosine value of a Given double value.
Math.tan()	It is used to return the trigonometric Tangent value of a Given double value.
Math.asin()	It is used to return the trigonometric Arc Sine value of a Given double value
Math.acos()	It is used to return the trigonometric Arc Cosine value of a Given double value.
Math.atan()	It is used to return the trigonometric Arc Tangent value of a Given double value.

Angular Math Methods

Method Name	Description
Math.toDegrees	It is used to convert the specified Radians angle to an equivalent angle measured in Degrees.
Math.toRadians	It is used to convert the specified Degrees angle to an equivalent angle measured in Radians.

Some of the listed methods are briefly described in the following.

1. **abs()**

   The abs() method returns the absolute value of any type of argument i.e. int, long, float, or double.

- The result is positive zero if the argument is positive zero or negative zero.
- The result is positive infinity if the argument is infinite.
- The result is NaN (Not a Number) if the passed argument is NaN.

Syntax:

```
public static datatype abs(datatype arg)
```

For example, `x = Math.abs(99); // output is 99`

```
x = Math.abs(-99) // output is 99 l
```

2. **ceil()**

   The ceil() method returns the smallest possible value that is greater than or equal to the argument and equal to the nearest

integer value. In other words, the argument is rounded up to the nearest integer equivalent.

- The result is the same if the returned value is already a mathematical integer.
- The result is the same if the passed argument is NaN or infinite or zero.
- The result is negative zero if the passed argument is less than zero but greater than -1.0

Syntax:

```
public static double ceil(double arg);
```

All the following calls to Math.ceil() return the double value 9.0:

```
Math.ceil(9.0) // result is 9.0
Math.ceil(8.8) // rises to 9.0
Math.ceil(8.02) // still rises to 9.0
Math.ceil(-9.0) //return -9.0
```

3. floor()

The floor() method returns the closest integer value that is less than or equal to the argument. This method is the reverse of the ceil() method.

Syntax:

```
public static double floor(double a);
```

All the following calls to Math.floor() return the double value 9.0:

```
Math.floor(9.0) // result is 9.0
Math.floor(9.4) // drops to 9.0
Math.floor(9.8) // still drops to 9.0
```

4. max()

The max() method takes two numeric arguments and returns the greater of the two. This method is overloaded to handle int, long, float, or double arguments. If the input parameters are the same, max() returns a value equal to the two arguments.

Syntax:

```
public static datatypemax(arg1, arg2);
```

For example: `s=Math.max(23,60)//returns 60`

`S= Math.max(56.54, 25.0) // returns 56.54`

5. min()

The min() method is the opposite of the max() method. It takes two numeric arguments and returns the lesser of the two. This method is overloaded to handle int, long, float, or double arguments. If the input parameters are the same, min() returns a value equal to the two arguments.

Syntax:

```
public static datatypemin(datatypearg1, datatype
arg2);
```

For example, `x = Math.min(0.5, 0.0); // output is 0.0`

6. random()

The random() method returns a random double that is greater than or equal to 0.0 and less than 1.0. The random() method does not take any parameters.

Syntax:

```
public static double random( );
```

7. round()

The round() method returns the integer closest to the argument. The algorithm is to add 0.5 to the argument and truncate to the nearest integer equivalent. This method is overloaded to handle a float or a double argument.

Syntax:

```
public static intround(float a);
public static long round(double a);
```

8. sin()

The sin() method returns the sine of an angle. The argument is a double representing an angle in radians. Degrees can be converted to radians by using Math.toRadians().

Syntax:

```
public static double sin(double a);
```

For example, `Math.sin(Math.toRadians(90.0))// returns 1.0`

9. cos()

The cos() method returns the cosine of an angle. The argument is a double representing an angle in radians.

Syntax:

```
public static double cos(double a);
```

For example, `Math.cos(Math.toRadians(0.0)) // returns 1.0.`

10. tan()

The tan() method returns the tangent of an angle. The argument is a double representing an angle in radians.

Syntax:

```
public static double tan(double a);
```

For example, `Math.tan(Math.toRadians(45.0)) // returns 1.0.`

11. sqrt()

The sqrt() method returns the square root of a number in double. This square root function returns a complex number (comprised of real and imaginary parts) when the operand is negative.

The Java Math.sqrt() method returns NaN instead of an object representing a complex number.

Syntax:

```
public static double sqrt(double a);
```

For example, `Math.sqrt(9.0) // returns 3.0`

12. toRadians()

This method converts argument (degree) to radians.

Syntax:

```
public static double toRadians(double deg);
```

13. log()

This method returns the logarithmic value of the passed argument.

Syntax:

```
public static double log(double arg);
```

14. atan()

This method returns the arc tangent of the argument value. The returned angle is in the range -pi/2 through pi/2. arc tan is inverse tan of the argument passed.

Syntax:

```
public static double atan(double a);
```

15. asin()

This method returns the arc sine value of the argument passed. The returned angle is in the range -pi/2 to pi/2. arc sine is inverse sine of the argument passed.

Syntax:

```
public static double asin(double arg);
```

16. cbrt()

This method returns the cube root of the passed argument.

Syntax:

```
public static double cbrt(double arg);
```

9.3.2 Program Examples

The application of the above methods is shown in the following examples.

Example 9.1: Program showing the use of java math classes like max, sqrt, pow, log, and exp

```
public class JavaMathExample1
{
public static void main(String[] args)
{
double x = 60;
double y = 3;
// return the maximum of two numbers
System.out.println("Maximum number of x and y is:
```

```
" +Math.max(x, y));
// return the square root of y
System.out.println("Square root of y is: " +
Math.sqrt(y));
//returns x to power of 3
System.out.println("Power of x and y is: " +
Math.pow(x, y));
// return the logarithm of given value
System.out.println("Logarithm of x is: " +
Math.log(x));
System.out.println("Logarithm of y is: " +
Math.log(y));
// return the logarithm of given value when base
is 10
System.out.println("log10 of x is: " +
Math.log10(x));
System.out.println("log10 of y is: " +
Math.log10(y));
// return the log of x + 1
System.out.println("log1p of x is: "
+Math.log1p(x));
// return a power of 2
System.out.println("exp of a is: " +Math.exp(x));
// return (a power of 2)-1
System.out.println("expm1 of a is: "
+Math.expm1(x));
}
}
```

Example 9.2: Program showing the use of java math class like sin, cos, tanh

```
public class JavaMathExample2
{
```

```
public static void main(String[] args)
{
 double a = 30;
 // converting values to radian
  double b = Math.toRadians(a);
// return the trigonometric sine of a
   System.out.println("Sine value of a is: "
+Math.sin(a));
 // return the trigonometric cosine value of a
   System.out.println("Cosine value of a is: "
+Math.cos(a));
 // return the trigonometric tangent value of a
  System.out.println("Tangent value of a is: "
+Math.tan(a));
  // return the trigonometric arc sine of a
   System.out.println("Sine value of a is: "
+Math.asin(a));
 // return the trigonometric arc cosine value of a
  System.out.println("Cosine value of a is: "
+Math.acos(a));
 // return the trigonometric arc tangent value of
a
  System.out.println("Tangent value of a is: "
+Math.atan(a));
// return the hyperbolic sine of a
      System.out.println("Sine value of a is: "
+Math.sinh(a));
// return the hyperbolic cosine value of a
  System.out.println("Cosine value of a is: "
+Math.cosh(a));
// return the hyperbolic tangent value of a
System.out.println("Tangent value of a is: "
+Math.tanh(a));
```

```
}
}
```

Example 9.3: Program showing use of java math class like abs(), acos(), toRadians

```
// Java program explaining lang.Math class methods
// abs(), acos(), toRadians()
import java.lang.*;
public class NewClass
{
    public static void main(String[] args)
    {
        // Declaring the variables
        int Vali = -1;
        float Valf = .5f;

        // Printing the values
         System.out.println("Initial value of int
: "+Vali);
         System.out.println("Initial value of int
: "+Valf);
 // Use of .abs() method to get the absoluteValue
        int Absi = Math.abs(Vali);
        float Absf = Math.abs(Valf);
        System.out.println("Absolute value of int
: "+Absi);
       System.out.println("Absolute value of int
: "+Absf);
        System.out.println("");
         // Use of acos() method
        // Value greater than 1, so passing NaN
        double Acosi = Math.acos(60);
       System.out.println("acos value of Acosi :
```

```
"+Acosi);
        double x = Math.PI;
       // Use of toRadian() method
        x = Math.toRadians(x);
        double Acosj = Math.acos(x);
       System.out.println("acos value of Acosj :
"+Acosj);
    }
}
```

Example 9.4: Program showing use of java math class like asin() and cbrt()

```
// Java program explaining lang.Math class methods
asin(), cbrt()

import java.lang.*;
public class NewClass
{
public static void main(String[] args)
{
int a = 1, b = 8;
int radd = a+b;
       // Use of asin() method
        // Value greater than 1, so passing NaN
        double Asini = Math.asin(radd);
System.out.println("asin value of Asini : " +
Asini);
        double x = Math.PI;
       // Use of toRadian() method
        x = Math.toRadians(x);
        double Asinj = Math.asin(x);
System.out.println("asin value of Asinj : " +
Asinj);
```

```
System.out.println("");
        // Use of cbrt() method
        double cbrtval = Math.cbrt(216);
System.out.println("cube root : " + cbrtval);
    }
}
```



CHECK YOUR PROGRESS

Q 1. Choose the correct options from the followings:

i) Which of these classes is a superclass of all other classes?

a) Math b) Process c) System d) Object

ii) What is the value of double constant 'E' defined in Math class?

a) approximately 3 b) approximately 3.14

c) approximately 2.72 d) approximately 0

iii) Which of these classes contains only floating point functions?

a) Math b) Process c) System d) Object

iv) What will be the output of the following Java program?

```
  class Main
{
public static void main(String args[])
{
double a=3.1;
double b=6.7;
double c=Math.max(a,b);
System.out.print (c);
    }
}
```

a) 4.5 b) 6.7 c) 3.1 d) None of these

v) What is the value of "d" in the following Java code snippet?

double m=Math.round(2.5 + Math.random());

a) 2 b) 3 c) 4 d) 2.5

vi) Which of the following would compile without error?

a) int a = Math.abs(-5); b) int a = Math.abs(5.0);

c) int a = Math.abs(5.5F); d) int a = Math.abs(5L);

vii) Which of the following are valid calls to Math.max?

1.Math.max(2,7) 2. Math.max(4.5,7)

3. Math.max(2,3,5) 4. Math.max(-4.5f,-9.0f)

a) 1,2,3 b) 2,3,4 c) 1,2,4 d) 3,4

viii) To declare a constant MAX_LENGTH as a member of the class, we have to write as:

a) final static MAX_LENGTH = 99.98;

b) final static float MAX_LENGTH = 99.98;

c) static double MAX_LENGTH = 99.98;

d) final static double MAX_LENGTH = 99.98;



9.4 LET US SUM UP

- The java.lang package defines classes that are fundamental to the Java language.
- All classes in the java.lang package are imported automatically, so there is no reason to write an import statement for them.
- The java.lang package also contains the Math class, which is used to perform basic mathematical operations.
- Java.lang.Math class provides several methods to work on mathematical operations.
- Various methods under Math class include min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs(), etc.
- The methods of the Math class are static and are accessed like any static method-through the class name. For these method calls, the general form is: res = Math.aStaticMathMethod();



9.5 FURTHER READING

1) Schildt, H. (2014). Java: The Complete Reference, Oracle Press.

2) Rao, R. N. (2006). Core and Advanced Java Black Book, Dreamtech Press.

9.6 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. i) d ii) c iii) a iv) b v) b vi) b vii) c viii) d

9.7 MODEL QUESTIONS

Q 1. What are the constants available in java.lang.Math class? Explain them with example.

Q 2. Write the usability of java.lang.Math class.

Q 3. Explain various methods of java.lang.Math class which are inherited from Object class.

Q 4. Write a Java program to demonstrate the use of three trigonometric functions.

*** ***** ***

UNIT 10: EXCEPTION HANDLING

UNIT STRUCTURE

10.1 Learning Objectives

10.2 Introduction

10.3 Exception Handling Fundamentals

10.3.1 Exception

10.3.2 Exception Types

10.4 Exception Handling Mechanism

10.5 Built-in and User-Defined Exception

10.6 Let Us Sum Up

10.7 Further Reading

10.8 Answer to Check Your Progress

10.9 Model Questions

10.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe the concept of exception and its fundamentals
- describe the process of handling exceptions in the Java language
- use built-in and user-defined exceptions.

10.2 INTRODUCTION

Exception handling in Java is one of the powerful mechanisms to handle runtime errors so that the normal flow of the application can be maintained. In the last unit, we discussed java.lang.Math class and the various methods under this class. In this unit, you will learn about java exception, their various types, and their differences. Further, you will get the knowledge of how to handle the exception. In the next unit, we will explore topics related to file handling.

10.3 EXCEPTION HANDLING FUNDAMENTALS

10.3.1 Exception

According to Oxford Dictionary, exception means an abnormal condition. In Java, exception refers to an unexpected event that disrupts the normal flow of the program. An exception may occur during the execution of a program(at run time), to disrupt the normal flow of the program's instructions. It leads to the abnormal termination of the program, which is not always recommended. In Object Oriented languages like Java, exceptions are known as objects. They are used to pass information about unusual events to and from the different parts of the program.

An exception may occur due to the following reasons.

- Invalid user data input.
- Network connection may be disturbed in the middle of communications
- JVM may run out of memory.
- File cannot be found/opened.

Exception Hierarchy:

The java.lang.The exception class is the base class for all exception classes. Figure 1 presents the exception hierarchy. All exception and error types are subclasses of the class Throwable, which is the base class of the exception hierarchy. One branch is headed by Exception. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. In another branch, Errors are used by the Java virtual machine (JVM) to indicate errors having to do with the runtime environment itself(JRE). StackOverflowError is an example of such an error. Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated by the runtime environment.JVM out of memory is an example of an Error. Normally, programs cannot recover from errors.

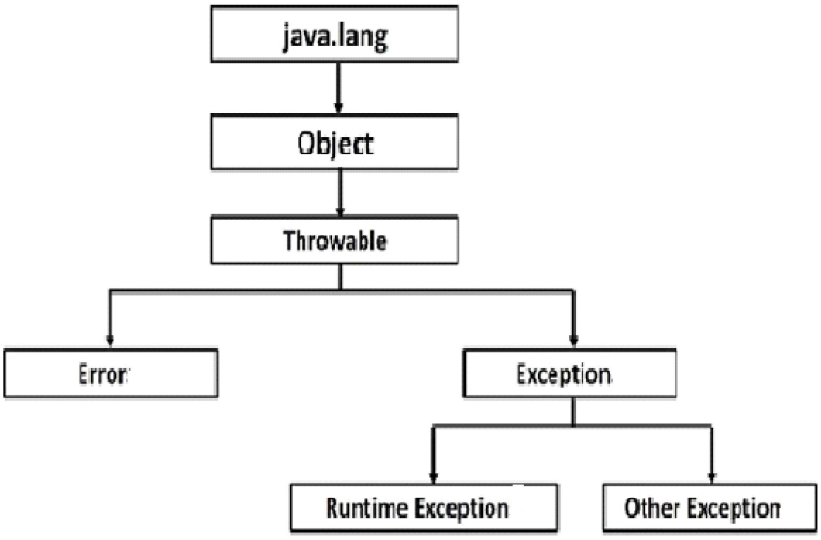


Figure10. 1: Exception Hierarchy

10.3.2 Exception Types

There are mainly three types of exceptions:

- Checked Exception
- Unchecked Exception
- Error
- **Checked Exception:**

These types of exceptions are checked by the compiler during the compilation to confirm whether the exception is handled by the programmer or not. The classes that extend the Throwable class except RuntimeException and Error are known as checked exceptions. Some checked exceptions include IOException, SQLException, etc. If some code within a method throws a checked exception, then the method must either handle the exception by using the throws keyword.

- **Unchecked Exception:**

Unchecked exceptions are not checked at compile-time rather they are checked at runtime. The classes that extend RuntimeException are known as unchecked exceptions such as ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.

- **Error:**

Errors are not exceptions, but these arise when the problem solution goes beyond the control of the user or the programmer. Fundamentally, an Error is used by the Java run-time system (JVM) to indicate errors that are associated with the run-time environment (JRE). Error is irretrievable. Some Errors include OutOfMemoryError, VirtualMachineError, AssertionError, etc.

10.4 EXCEPTION HANDLING MECHANISM

Exception Handling is a mechanism to deal with runtime errors such as ClassNotFound, ArrayIndexOutOfBound, etc. An exception is thrown when an error situation occurs during program execution. It is propagated by the JVM and caught by an exception handler that takes appropriate action to handle the situation. The core advantage of exception handling is to maintain the normal flow of the execution of an application. Exception normally disrupts the normal flow of the application due to which we use an exception handling mechanism. The exception handling mechanism works in a flow of execution which is depicted in figure2. If an exception is not handled at the proper time, it may lead to a system failure. That is why handling an exception is very considered important.



Figure 10.2. Workflow of Exception Handling

In Java, the following keywords are used for exception handling.

- try
- catch
- finally
- throw

- throws

Throwing and catching exceptions:

Catching Exceptions

A method catches an exception using a combination of the try and catch keywords. The program code that may generate an exception should be placed inside the try-catch block.

The try-catch block is used as follows:

```
try {
// block of code to monitor for errors
}
catch(ExceptionType1exOb)
{
// exception handler for ExceptionType1
}
catch(ExceptionType2exOb)
{
// exception handler for ExceptionType2
}
//optional
finally {
// block of code to be executed after try block
ends
}
```

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception is handled by the catch block associated with it. Every try block should be immediately followed either by a catch block or a final block.

A catch statement involves declaring the type of exception which is tried to catch. If an exception occurs, then the catch block (or blocks) which follow the try block is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block similar to an argument that is passed into a method parameter.

```
classException_example{
publicstaticvoidmain(Stringargs[])
{
inta,b;
try { // monitor a block of code.
a = 0;
b = 10 / a; //raises the arithmetic exception
System.out.println("Tryblock.");
}
catch (ArithmeticExceptione)
{    //    catch    divide-by-zero    error
System.out.println("Exception found:Divisionbyzero.");
}
System.out.println("Aftertry/catchblock.");
}
}
```

Output:

Exception found: Division by zero.

Aftertry/catch block.

Multiple catch clauses:

In some cases, more than one exception could be raised by a single piece of code. To handle this multiple exception, two or more catch clauses can be specified. Here, each catch block catches different types of exceptions. When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed. After one catch statement executes, the others are by passed, and execution continues after the try/catch block. The following example traps two different exception types:

```
classMultiCatch{
publicstaticvoidmain(Stringargs[]){
try{
int x, y;
```

```
x = args.length; System.out.println("x="+x);
y=10/x;//maycause division-by-zeroerror
intarr[] = { 10,20 };
c[5] =100;
}
catch(ArithmeticExceptione)
{
System.out.println("Divideby0:"+ e);
}
catch(ArrayIndexOutOfBoundsExceptione)
{
System.out.println("Arrayindexoob:"+e);
}
System.out.println("Aftertry/catchblocks.");
}
}
```

Outputs generated by theprogram when executed in both ways:

C:\>javaMultiCatch

x = 0

Divideby0:java.lang.ArithmeticException:/byzero

After try/catch blocks.

C:\>javaMultiCatcharg1

x = 1

Array index oob:java.lang.ArrayIndexOutOfBoundsException:5

After try/catch blocks.

While multiple catch statements are used, it is important to remember that exception subclasses must come before their superclasses. A catch statement that uses a superclass will catch exceptions of that type plus any of its subclasses. Thus, a subclass would never be reached fit came after its superclass. Moreover, unreachable code in Java encounters an error. For example, consider the following program:

```
classMultiCatch{
publicstaticvoidmain(Stringargs[]){
try{
inta,b;
a = args.length; System.out.println("a="+a);
b=10/a;//maycausedivision-by-zeroerror intarr[] =
{ 10,20 };
c[5] =100;
}

catch(Exception e) {
System.out.println("GenericExceptioncatch.");
}
catch(ArithmeticExceptione)
{
System.out.println("Divideby0:"+ e);
}
catch(ArrayIndexOutOfBoundsExceptione)
{
System.out.println("Arrayindexoob:"+e);
}
      System.out.println("Aftertry/catchblocks.");
}
}
```

Nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested in the following way.

```
try
{
statement1;
```

```
statement2;
try
{
statement1;
statement2;
}
catch(Exceptione)
{
}
}
catch(Exceptione){}....
```

Throw keyword

The throw keyword is used to explicitly throw an exception. The general form of throw is:

```
throwThrowableInstance;
```

Here, Throwable Instance must be an object of type Throwable class or a subclass of Throwable. Primitive types,such as int or char,as well as nonThrow able classes, such as String and Object, cannot be used as exceptions.

There are two ways to obtain a Throwable object:

- using a parameter in a catch clause.
- creating one with the new operator.

The following program explains the use of throw keyword.

```
public class TestThrow1
{ staticvoidvalidate(intage){
try{
if(age<18)
thrownewArithmeticException("notvalid");
else
System.out.println("Eligibletovote");
}
catch(ArithmeticExceptione)
```

```
{
System.out.println("CaughtinsideArithmeticExceptions.");
throw e; // rethrow the exception
}
}
publicstaticvoidmain(Stringargs[]){
try{
validate(13);
}
catch(ArithmeticExceptione)
{
System.out.println("ReCaughtArithmeticExceptions.");
}
}
}
```

The flow of execution stops immediately after the throw statement and any subsequent statements that are not executed.The nearest try block is employed to see if it has a catch statement that matches the type of exception. If it finds a match, control is trans ferred to that statement. If not, then the next enclosing try statement is inspected, and so on. If no matching catch is found, then the default exception handle rhalts the programand prints the stack trace.

Throws keyword

The throws keyword is used to specify that a method throws an exception but does not handle it, it leaves it up to the caller of the method to handle the exception. In such case, the throws keyword can be specified in the method declaration along with the name of the exception that the method throws. If a method throws a checked exception, then it must either be caught and handled by the method or declared in the throws clause.Otherwise, a compilation error occurs. Unchecked exceptions need not be specified in the throws clause.

For example:

```
public class LoadClass {
public void load Class (String class Name) throws
Class Not Found Exception
{
  Class.forName(className);
 }
}

public class Main {
 public static void main(String[] args) {
  LoadClasslc = new LoadClass();
  try {
   lc.loadClass("com.MissingClass");
  }
catch (ClassNotFoundException e)
{
      System.out.println("This class is
notpresent:"+e.getMessage());
  }
}
}
```

Output:

This class is not present:com.MissingClass

The Finally Block

The final block follows a try block or a catch block. A final block of code always executes, irrespective of the occurrence of an Exception. A final block appears at the end of the catch blocks that follow the syntax as given below.

Syntax:

```
try {
//Protected code
```

```
}catch(ExceptionType1e1){// Catch block}
catch(ExceptionType2e2){// Catch block}
finally{
//Thefinallyblockalwaysexecutes.
}
```

Example:

```
publicclassFinally_Example{
publicstaticvoidmain(Stringargs[]){
try {
inta,b;
a=0;
b=10/a;
}
catch (ArithmeticException e) {
System.out.println("Exceptionthrown:"+e);
}
finally{
System.out.println("Thefinallyblockisexecuted");
}
}
}
```

Points to remember:

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.·

  The try block cannot be present withoute it her catch clause or finally clause.

  Any code cannot be present in between the try, catch, finally blocks.

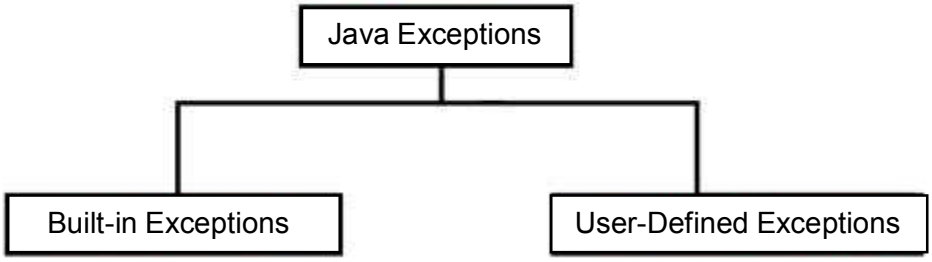
  Keywords used for exception handling-
- try block contains program statements that are to be to monitored for exceptions. If an exception occurs within the try block, it is thrown.
- catch block contain statements that catch this exception and handle it in some rational manner – System-generated exceptions are

automatically thrown by the Java runtime system.

- throw keyword is used to manually throw an exception.
- throws keyword is used to specify any exception that is thrown out of a method.
- finally block contains code that must be executed after a try block completes.

10.5 BUILT-IN AND USER DEFINED EXCEPTIONS

Java defines several types of built-in exceptions that relate to its various class libraries. Also, Java allows users to define their own exceptions.



Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations. Some important built-in exceptions in Java are stated in the following. catch block contain statements that catch this exception and handle it.

- **ArithmeticException:** It is thrown when an exceptional condition has occurred in an arithmetic operation.
- **ArrayIndexOutOfBoundsException:** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
- **ClassNotFoundException:** This Exception is raised when we try to access a class whose definition is not found.
- **FileNotFoundException:** This Exception is raised when a file is not accessible or does not open.
- **IOException:** It is thrown when an input-output operation failed or interrupted.
- **InterruptedException:** It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

- **NoSuchFieldException:** It is thrown when a class does not contain the field (or variable) specified.
- **NoSuchMethodException:** It is thrown when accessing a method that is not found.
- **NullPointerException:** This exception is raised when referring to the members of a null object. Null represents nothing.
- **NumberFormatException:** This exception is raised when a method could not convert a string into a numeric format.
- **RuntimeException:** This represents an exception that occurs during runtime.
- **StringIndexOutOfBoundsException:** It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string.
- **IllegalArgumentException:** This exception will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.
- **IllegalStateException:** This exception will throw an error or error message when the method is not accessed for the particular operation in the application. It belongs to the unchecked exception.

Examples of using some Built-in Exceptions:

Example 10.1: Java program to demonstrate Aritmetic Exception

```
// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo{
public static void main(String args[])
{
    try{
        int a = 30, b = 0;
        int c = a/b; // cannot divide by zero
        System.out.println ("Result = "+ c);
    }
```

```
        catch(ArithmeticException e) {
            System.out.println ("Can't divide a
number by 0");
        }
    }
}
```

Output

Can't divide a number by 0

Example 10.2: Java program to demonstrate **StringIndexOutOfBoundException**

```
// Java program to demonstrate
//StringIndexOutOfBoundsException
class StringIndexOutOfBound_Demo
{
    public static void main(String args[])
    {
        try {
         String a = "This is like chipping ";
// length is 22
    char c = a.charAt(24); // accessing 25th element
System.out.println(c);
        }
catch(StringIndexOutOfBoundsException e) {
System.out.println("StringIndexOutOfBoundsException");
        }
    }
}
```

Output : StringIndexOutOfBoundsException

Example 10.3 Java program to demonstrate **ArrayIndexOutOfBoundsException**

```
// Java program to demonstrate ArrayIndexOutOfBoundException
class ArrayIndexOutOfBound_Demo
{
    public static void main(String args[])
    {
        try{
            int a[] = new int[5];
            a[6] = 9; // accessing 7th element in an
// array of size 5
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println ("Array Index is Out
Of Bounds");
        }
    }
}
```

Output

Array Index is Out Of Bounds

User-Defined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, the user can create exceptions which are called 'user-defined Exceptions'. Java allows the user to create their exception class which is derived from the built-in class Exception. The user-defined exception can be thrown using the throw keyword.

Syntax to define a user-defined exception:

```
class User_defined_name extends Exception
  {
            ……..
}
```

- We can write a default constructor in its own exception class.

```
MyException(){}
```

- We can also create a parameterized constructor with a string as a parameter. We can use this to store exception details. We can call the superclass(Exception) constructor from this and send a string.

```
MyException(String str)
{  super(str);}
```

- To raise an exception of a user-defined type, we need to create an object to his exception class and throw it using the throw clauseas given here:

```
MyException me = new MyException("Exception
details");
        throw me;
```

The following program explains how to create your own exception class MyException.

Example 10.4: Java program to demonstrate a user-defined exception. This program throws an exception whenever the balance amount is below Rs 1000

```
class MyException extends Exception
{
    //store account information
    private static int accno[] = {1001, 1002, 1003,
1004};
    private static String name[] =
            {"Nish", "Shubh", "Sush", "Abhi",
"Akash"};
    private static double bal[] =
         {10000.00, 12000.00, 5600.0, 999.00,
1100.55};
    // default constructor
    MyException() {    }
     // parameterized constructor
    MyException(String str) { super(str); }
```

```
        // write main()
    public static void main(String[] args)
    {
        try  {
            // display the heading for the table
            System.out.println("ACCNO"+ "\t"+
"CUSTOMER"+
         "\t"+ "BALANCE");

            // display the actual account information
            for(int i = 0; i<5;i++)
            {
                System.out.println(accno[i] + "\t"+
name[i] +
          "\t"+ bal[i]);

          // display own exception if balance < 1000
                if(bal[i] < 1000)
                {
                MyException me =
           new MyException("Balance is less than
1000");
                    throw me;
                }
            }
        } //end of try

        catch(MyException e) {
            e.printStackTrace();
        }
    }
}
```

Output: Runtime Error

MyException: Balance is less than 1000

at MyException.main(fileProperty.java:36)



CHECK YOUR PROGRESS

Q 1. When does Exceptions in Java arises in code sequence?

a) Run Time b) Compilation Time c) Can Occur Any Time

d) None of the mentioned

Q 2. Which of these keywords is not a part of exception handling?

a) try b) finally c) thrown d) catch

Q 3. Which of these keywords must be used to handle the exception thrown by try block in some rational manner?

a) try b) finally c) throw d) catch

Q 4. The class at the top of exception class hierarchy is _____

a) ArithmeticException b) Throwable c) Object

d) Exception

Q 5. Which keyword is used to explicitly throw an exception?

a) try b) throwable c) catch d) throw

Q 6. Which exception is thrown when divide by zero executes?

a) NumberFormatException b) ArithmeticException

c) NullPointerException d) None of these

Q 7. Consider the following program.

```
public class Test {
public static void main (String args[])
{  try  { int a=Integer.parseInt("One");  } }
}
```

Which exception to be handled by catch block in the above code?

a) IllegalStateException b) NumberFormatException

c) ArrayIndexOutOfBoundsException d) None of these

Q 8. What is the output of the following code with exception?

public class exception_test

```
{
public static void main(String args[]) throws
ArithmeticException
{
System.out.println ("We are the learners");
throw new  ArithmeticException();
}
}
```

a) We are the learners b) No output c) Compile time error
d) None of these

Q 9. Exception class is found in which package in java?

a) java.util b) java.io c) java.lang d) java.net

Q 10. Which block is always executed independently of an exception being raised?

a) throws b) finally c) throw d) catch



10.6 LET US SUM UP

- An exception refers to an unexpected event that disrupts the normal flow of the program.
- An exception may occur during the execution of a program (at run time), to disrupt the normal flow of the program's instructions. It leads to the abnormal termination of the program, which is not always recommended. In Object Oriented languages like Java, exceptions are known as objects.
- An exception may occur due to the following reasons.
  - Invalid user data input.
  - Network connection may be disturbed in the middle of communications.
  - JVM may run out of memory.
  - File cannot be found/opened.
- The java.lang.Exception class is the base class for all exception classes. All exception and errors types are sub classes of class Throwable, which

is the base class of exception hierarchy.

- There are mainly three types of exceptions: Checked Exception, Unchecked Exception and Error.
- Checked Exception are checked by the compiler during the compilation to confirm whether the exception is handled by the programmer or not.
- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.
- Unchecked Exception are not checked at compile-time rather they are checked at runtime. The classes that extend RuntimeException are known as unchecked exceptions such as ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- Errors are not actually exceptions, but these arise when problem solution goes beyond the control of the user or the programmer. Fundamentally, an Error is used by the Java run-time system (JVM) to indicate errors that are associated with the run-time environment (JRE). Error is irretrievable. Some Errors include Out Of Memory Error, Virtual Machine Error, Assertion Error, etc.
- Exception Handling is a mechanism to deal with runtime errors such as Class Not Found, Array Index Out Of Bound etc.
- An exception is thrown when an error situation occurs during program execution. It is propagated by the JVM and caught by an exception handler that takes an appropriate action to handle the situation.
- Java provides five keywords to handle the exception. Those are try, catch, throw, throws and finally.
- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause. Any code can not be presentin between the try, catch, finally blocks.
- Java defines several types of built-in exceptions that relate to its various class libraries. Further, Java also allows users to define their own exceptions.



10.7 FURTHER READING

1) Schildt, H. (2014). Java: The Complete Reference, Oracle Press.
2) Rao, R. N. (2006). Core and Advanced Java Black Book, Dreamtech Press



10.8 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. a

Ans to Q No 2. c

Ans to Q No 3. d

Ans to Q No 4. b

Ans to Q No 5. d

Ans to Q No 6. b

Ans to Q No 7. b

Ans to Q No 8. a

Ans to Q No 9. c

Ans to Q No 10. b



10.9 MODEL QUESTIONS

Q 1. What do you mean by exception? Discuss various types of exceptions.

Q 2. How can we handle exception? Name various keywords used in Java to handle the exception.

Q 3. What is the difference between exception and error?

Q 4. Describe the use of the throw keyword with an example.

Q 5. Is there any difference between throw and throws in exception handling in Java? If yes, discuss it with suitable example.

Q 6. Define user-defined or custom exceptions in Java with proper example.

*** ***** ***

UNIT 11: FILE HANDLING

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Java I/O

11.4 I/O Basics

11.4.1 Stream and Stream Classes

11.5 Reading Console Input

11.6 Writing Console Output

11.7 Reading and Writing Files

11.8 Let Us Sum Up

11.9 Further Reading

11.10 Answer to Check Your Progress

11.11 Model Questions

11.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- explain the concept of I/O (Input-Output)
- define stream
- explain the various streams for I/O
- describe the file and the different file operations
- explain reading and writing console input and console output
- describe how to read and write files.

11.2 INTRODUCTION

In the earlier unit, we learned about the concept of exception handling. We have learned about java exception, its various types and their differences, and how to handle the exception. In this unit, you will be introduced to the concepts related to file handling.

Topics like Java I/O, I/O basics, and different stream classes are discussed in detail in this unit. In addition to these, concepts related to

reading console input, writing console output, and reading and writing files along with different file operations are also described in this unit. In the next unit, we will explore topics related to Java applets.

11.3 JAVA I/O

Java I/O (Input-Output) is a standard mechanism that processes the input and generates the output. The package "java.io" contains the methods to perform all the input and output operations. To perform I/O operations faster, Java uses the concept of streams. A stream can be defined as a sequence of data consisting of bytes. A stream refers to a logical representation of the flow of data. In this unit, the background of I/O along with its related term Stream will be discussed. Further, the way of reading and writing files in Java will also be explained.

11.4 I/O BASICS

Programs read inputs from data sources (e.g., keyboard, file, network, memory buffer, or another program) and write outputs to a data sink (e.g., display console, file, network, memory buffer, or another program). In Java standard I/O, inputs and outputs are handled by the so-called streams. A stream is a sequential and logical representation of a one-way flow of data. It is important to mention that Java does not differentiate between the various types of data sources in the stream I/O. They are all treated as a sequential flow of data. Input and output streams can be established from/to any data source/sink, such as files, network, keyboard/console, or another program. The Java program receives data from a source by opening an input stream and sends data to a sink by opening an output stream. If a program needs to perform both input and output operations, we have to open two streams - an input stream and an output stream.

Stream I/O operations involve three steps:

1. Open an input/output stream associated with a physical device (e.g., file, network, console/keyboard), by constructing an appropriate I/O

stream instance.

2. Read from the opened input stream until "end-of-stream" is encountered, or write to the opened output stream (and optionally flush the buffered output).
3. Close the input/output stream.

Java internally stores characters (char type) in the 16-bit UCS-2 character set. But the external data source/sink could store characters in another character set (e.g., US-ASCII, ISO-8859-x, UTF-8, UTF-16, and many others), in a fixed length of 8-bit or 16-bit, or variable length of 1 to 4 bytes.

On the other hand, in Java, a File is known as a named location used to store related information. There are several File Operations like creating a new File, getting information about a File, writing into a File, reading from a File, and deleting a File.

To understand the File operations, it is required to know the Stream.

11.4.1 Stream and Stream Classes

A stream is simply a logical representation of a sequence of data. It is also defined as a sequence of bytes. It acts as an object that either delivers data to its destination or that takes data from a source. The source and destination can be disk files, arrays, text files, etc. The input stream reads or retrieves data from a source, whereas the output stream writes data to a destination. Based on the data type, there are two types of streams: Byte Stream and Character Stream.

Byte Stream:

Byte Stream is mainly involved with byte data. A file-handling process with a byte stream is a process in which input is provided and executed with the byte data. Byte streams perform input and output of 8-bit bytes. They read and write data one byte at a time. Other stream types are built on top of byte streams.

Character Stream:

Character Stream is mainly involved with character data. A file-handling process with a character stream is a process in which an input is processed with the character data.

Figure 1 depicts these two types of streams along with basic classes under each category.

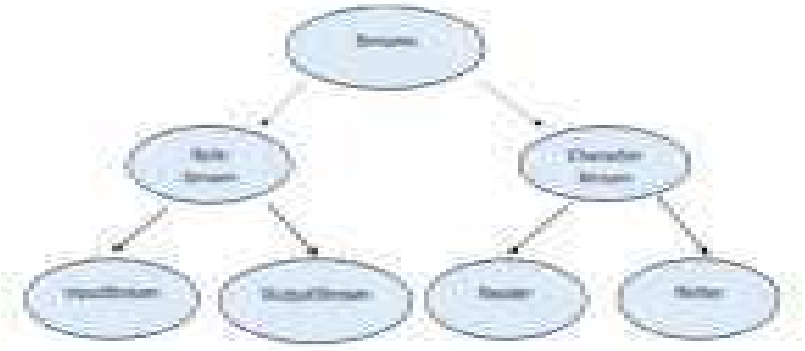


Figure11. 1: Classification of I/O streams based on data type

Again, depending on the type of operations, byte streams can be classified into two main groups: InputStream and OutputStream.

InputStream: This stream is used to read data that must be taken as an input from a source array or a file or any peripheral device. For example, FileInputStream, BufferedInputStream, ByteArrayInputStream, etc. are some predefined input stream classes as shown in figure 2.

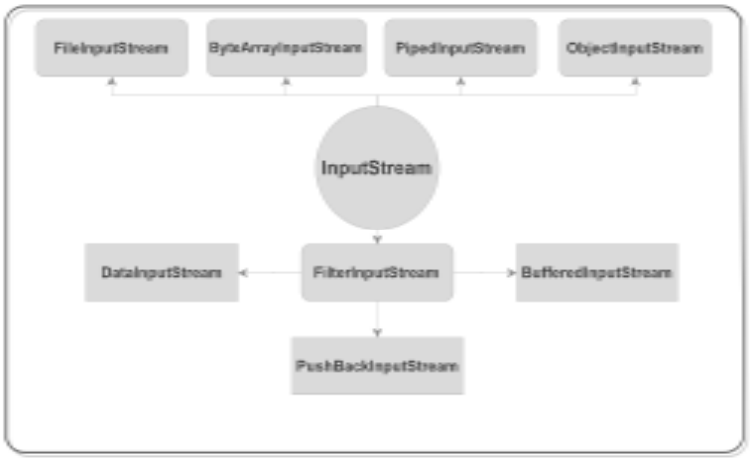


Figure11. 2: Classes under InputStream

OutputStream: This stream is used to write data as outputs into an array or file or any output peripheral device. For example, FileOutputStream, BufferedOutputStream, ByteArrayOutputStream etc. are predefined streams under this group. Figure 3 presents all the built-in classes under the OutputStream group.



Figure11. 3: Classes under OutputStream

Java brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all the types of objects, data types, characters, files, etc. to execute the I/O operations. The read operation and write operation through Java application are depicted in figure 4.



Figure11. 4: The flow of read-and-write operations

Java I/O stream is the flow of data that you can either read from, or you can write to. It is used to perform read and write operations in files permanently. Java uses streams to perform these tasks. Java I/O stream is also called File Handling, or File I/O. This Java I/O stream is available in the java.io package. Java.io package provides classes for system input and output through files, network

streams, memory buffers, etc.

Example 11.1: Program to read contents of a file

```
import java.io.*;
class ReadHello{
     public static void main(String args[])
{
        try    {
          FileInputStream    fin=new
FileInputStream("hello.txt");
          inti=0;
          while((i=fin.read())!=-1)  {
          System.out.println((char)i);
          }
          fin.close();
          }
        catch(Exception e)  {
          System.out.println(e);  }
               }
          }
     }
```

Example 11.2: Program to read from a file

```
import java.io.*;
     public class ReadFile {
public static void main(String[] args) {
BufferedReader br1 = null;
BufferedReader br2 = null;
    try    {
     br1  =  new  BufferedReader(new
FileReader("D:\\myfile.txt"));
     String content = br1.readLine();
     while (content != null) {
          System.out.println(content);
```

```
                content = br1.readLine();
      }
            }
      catch (IOExceptionioe)
      {
      ioe.printStackTrace();
      } }
            }
```

Example 11.3: Program in Java to write in a file

```
      import java.io.*;
            public class Hello{
            public static void main(String args[]){
             try  {
            F i l e O u t p u t s t r e a m f o = n e w
FileOutputStream("hello.txt");
            String i="Hello World ";
            byte  b[]=i.getBytes();/  /converting
string into byte array
            fo.write(i);
            fo.close();
                }
                    catch(Exception  e)     {
system.out.println(e);  }
               }
            }
      This will create a file hello.txt.
```

Output:

Hello World

Example 11.4: Program to write in a file

```
            import java.io.*;
            public class writeFirst     {
            public static void main(String args[])
```

```
            {
            File file=new File("visiters.txt");
            try   {
                  PrintWriter      output=new
PrintWriter(file);
                  output.println("Hello Visiters!
Welcome to Intellipaat");
                  output.close();
                        }
      catch(IOException        ex)        {
System.out.println("Error:"+ex);}
        }
      }
```

There are 3 standard or default streams System.in, System.out and System.err that Java provides for I/O operations which are shown in figure 5.

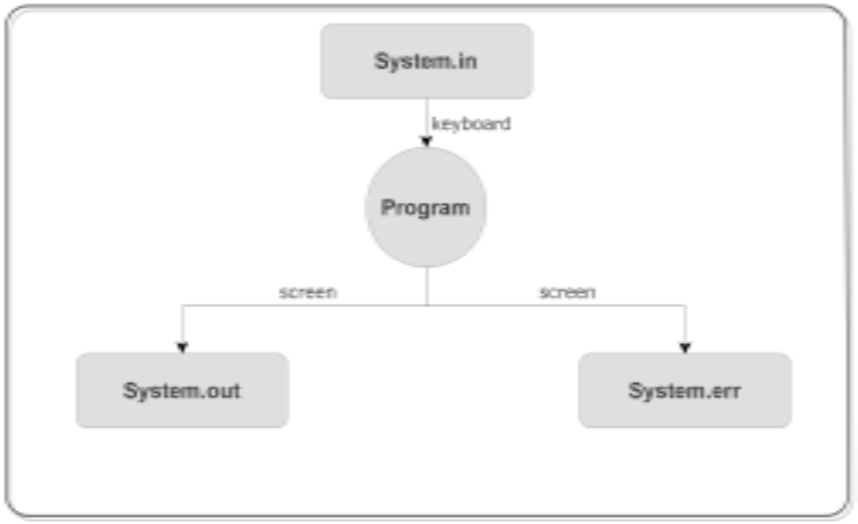


Figure11. 5: Standard Java streams

1. **System.in:** This is the standard input stream that is used to read characters from the keyboard or any other standard input device.
2. **System.out:** This is the standard output stream that is used to produce the result of a program on an output device like the

computer screen.

3. **System.err:** This is the standard error stream that is used to output all the error data that a program might throw, on a computer screen or any standard output device. This stream can also use all the 3 above-mentioned functions to output the error data.

The various print functions associated with System.out are discussed here.

print(): This method in Java is used to display a text on the console. The text is passed as the parameter to this method in the form of String. While displaying the text on the console, the cursor remains at the end of the text at the console.

Syntax: `System.out.print(parameter);`

Example 11.5: Program to show use of print()

```
// Java code to illustrate print()
import java.io.*;
class Demo_print {
    public static  void main(String[] args)
    {
        System.out.print("Welcome! ");
        System.out.print("Welcome! ");
        System.out.print("Welcome! ");
    }
}
```

Output:

Welcome! Welcome! Welcome!

println(): This method in Java is also used to display text on the console. However, it prints the text on the console, and then the cursor moves to the start of the next line at the console.

Syntax: `System.out.println(parameter);`

Example 11.6: Program to show the use of println()

```
// Java code to illustrate println()
import java.io.*;
```

```
class Demo_print {
    public static void main(String[] args)
    {
        System.out.println("Hello ");
        System.out.println("Hello ");
        System.out.println("Hello ");
    }
}
```

Output:

Hello
Hello
Hello

printf(): This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments. This is used to format the output in Java.

Example 11.7: Program to show working of printf()

```
// A Java program to demonstrate working of
// printf() in Java
class JavaFormat {
    public static void main(String args[])
    {        intx = 100;
System.out.printf("Printing simple"+"integer:x=%d\n",x);
  // this will print it upto 2 decimal places
System.out.printf("Formatted with"+"precision:PI = %.2f\n",
                  Math.PI);
                floatn = 5.2f;
// automatically appends zero to the rightmost
// part of decimal
System.out.printf("Formatted to "+ "specific width: n = %.4f\n",n);
```

```
                n = 2324435.3f;
      // here number is formatted from right margin
// and occupies 20 characters
       System.out.printf("Formatted to " + "right
margin: n = %20.4f\n", n);
          }
      }
```

Output:

Printing simple integer: x = 100

Formatted with precision: PI = 3.14

Formatted to specific width: n = 5.2000

Formatted to right margin: n = 2324435.2500

Stream Classification based on type of files: Depending on the types of files, Streams can be classified into two primary classes which can be further divided into other classes as shown in the figure 6.

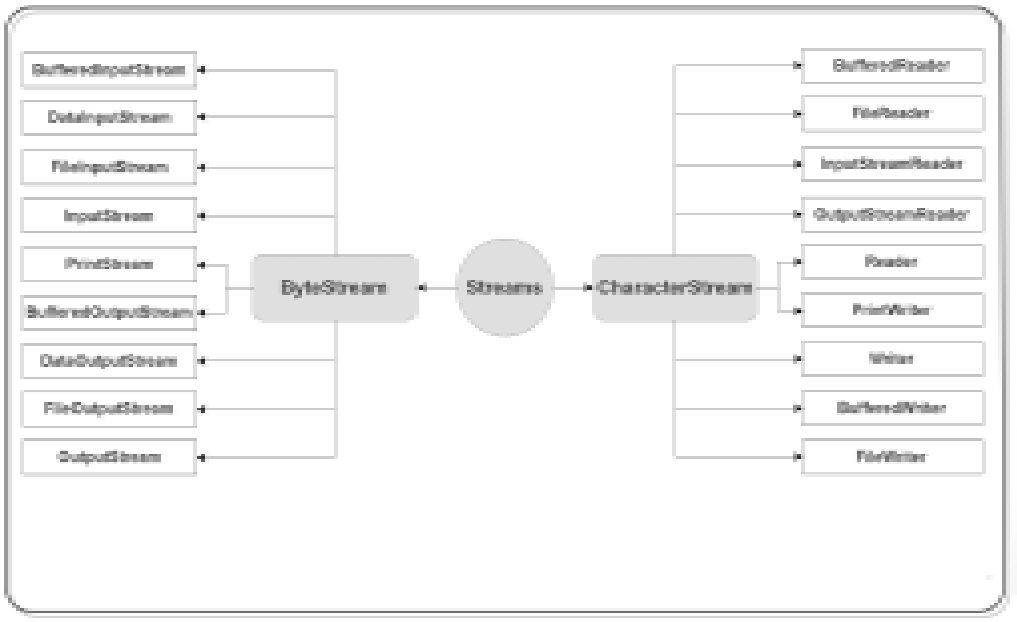


Figure 11. 6: Various classes under ByteStream and CharacterStream

ByteStream: This is used to process data byte by byte. Although ByteStream has many classes, the FileInputStream and the FileOutputStream are the mostly used classes. The FileInputStream

is used to read from the source and FileOutputStream is used to write to the destination. A list of various ByteStream Classes is given in Table 11.1.

Table 11.1: Various Classes under ByteStream

Stream class	Description
BufferedInputStream	It is used for Buffered Input Stream.
DataInputStream	It contains method for reading java standard datatypes.
FileInputStream	This is used to reads from a file
InputStream	This is an abstract class that describes stream input.
PrintStream	This contains the most used print() and println() method
BufferedOutputStream	This is used for Buffered Output Stream.
DataOutputStream	This contains method for writing java standard data types.
FileOutputStream	This is used to write to a file.
OutputStream	This is an abstract class that describe stream output.

Example 11.8: Program to show use of Byte Stream

```
// A Java Program to illustrating the Byte
Stream to copy the contents // of one file to
another file.
import java.io.*;
public class BYTEStream {
```

```
            publicstaticvoidmain( String[] args)
throwsIOException
          {
              FileInputStream sourceStream = null;
              FileOutputStreamtargetStream = null;
             try{
      source = newFileInputStream("File1.txt");
      target = new File Output Stream
("targetfile.txt");

      // Reading source file and writing content
to target file byte by byte
                  int temp;
                  while((temp = source.read()) !=
-1)
                      target.write((byte)temp);
              }
              finally{
                  if(source!=null)
                      source.close();
                  if(target!=null)
                     target.close();      }     }
}
```

CharacterStream: In Java, characters are stored using Unicode conventions where CharacterStream allows us to read/write data character by character. Among various classes, the FileReader and the FileWriter are the most common ones. FileReader reads from the source and FileWriter writes to the destination respectively. Table 2 presents a list of various CharacterStream Classes.

Table 11.2: Various CharacterStream Classes

Stream class	Description
BufferedReader	Used to handle buffered input stream.
FileReader	This is an input stream that reads from file.
InputStreamReader	Used to translate byte to character.
OutputStreamReader	This output stream is used to translate character to byte.
Reader	This is an abstract class that define character stream input.
PrintWriter	It contains the most used print() and println() method
Writer	This is an abstract class that define character stream output.
BufferedWriter	Used to handle buffered output stream.
FileWriter	Used to output stream that writes to file.

File Operations: The following operations on a file can be performed.

- Create a File
- Retrieve File Information
- Write to a File
- Read from a File
- Delete a File

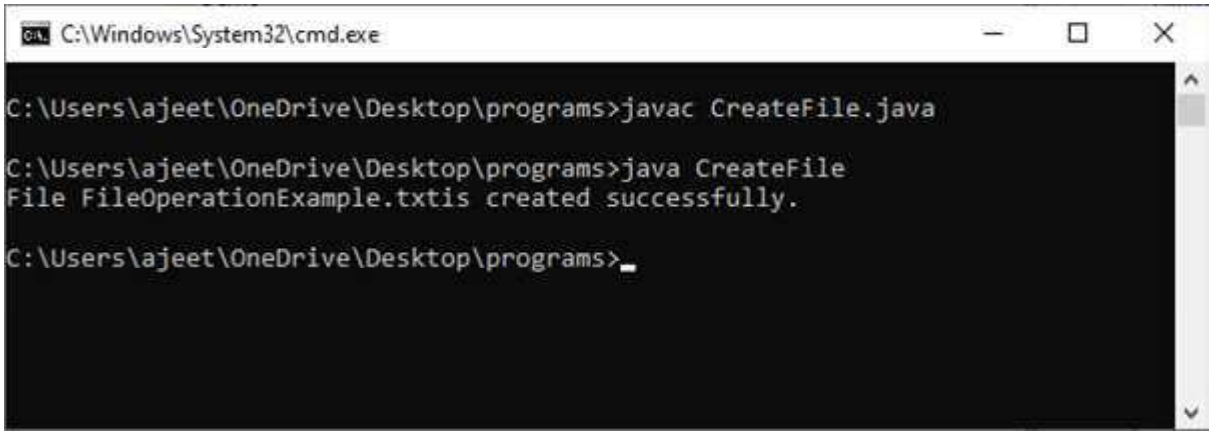
Create a File:

To create a File, we use the createNewFile() method of file. The createNewFile() method returns true if it successfully creates a new file and returns false when the file already exists. Following program creates a new file.

Example 11.9: Program to show how to create a file

```
// Importing File class
import java.io.File;
// Importing the IOException class for handling
errors
import java.io.IOException;
class CreateFile {
    public static void main(String args[]) {
        try {
    // Creating an object of a file
File f1 = new File("D:FileOperationExample.txt");
            if (f1.createNewFile()) {
System.out.println("File " + f1.getName() + " is
created successfully.");
       }
else {
 System.out.println("File is already exist in the
directory.");
                }
            }  catch (IOException exception) {
               System.out.println("An unexpected
error is occurred.");
                 exception.printStackTrace();
             }
       }
     }
```

Output:



Writing data to a file:

The write operation on a file means storing the data in a file. To perform write operations on a file, we have to use the write() method along with the FileWriter class. We should use the close() method to close the stream to retrieve the allocated resources.

Example 11.10: Program to write data to a file

```
importjava.io.*;
publicclassFileHandling
{
   publicstaticvoidmain(String []arr) throwsIOException
{
      // creating a new file and writing data to a file
      FileWriterfw=newFileWriter("demo.txt");
      String s="Welcome to Java File Handling.";
      fw.write(s);
      // closing a file
      fw.close();
    }
}
```

Reading from a file:

The read operation can access or retrieve the data stored from a file. To perform read operation on a file, we will use Scanner class along with has NextLine() and nextLine() methods to retrieve data

from a file. We should use the close() method to close the stream.

Example 11.11: Program to read data from a file

```
importjava.io.*;
importjava.util.Scanner;
publicclassFileHandle{
     publicstaticvoidmain(String []arr)
throwsIOException{
          File f=newFile("demo.txt");
          Scanner sc=newScanner(f);
          while(sc.hasNextLine())
          {
            String str=sc.nextLine();
            System.out.println(str);
          }
            sc.close();     }   }
```

Deleting a file:

In Java file handling, a file can be deleted using the delete() method of the File class.

There is no need to close the file using close() because neither the FileWriter nor the Scanner classes are used to delete a file.

Example 11.12: Program to delete a file

```
import java.io.*;
public class FileHandle  {
     public  static  void  main(String  []arr)
throwsIOException
    {
      File f=new File("DEMO.txt");
      Boolean result=f.delete();
      if(result)
         System.out.print(f+" deleted successfully.");
      else
```

```
        System.out.format("%s","File cannot be deleted
due to some error.");
   }
}
```

Retrieving file information:

There are few methods in Java that I have discussed earlier at methods of file class to get the information of the file.

Example 11.13: Program to retrieve file information file information

```
importjava.io.*;
publicclassFileHandle{
  publicstaticvoidmain(String []arr) throwsIOException{
     File file=newFile("demo.txt");
     file.createNewFile();
     String filename=file.getName();
     System.out.println("File Name is "+filename);
     System.out.println("Absolute path of "+filename+"
: "+file.getAbsolutePath());
       System.out.print("length of "+filename+" :
"+file.length());
       System.out.println("Is "+filename+" readable?
"+file.canRead());
       System.out.println("Is "+filename+" writable?
"+file.canWrite());
       System.out.println("Is "+filename+" exists?
"+file.exists());
  }
}
```

11.5 READING CONSOLE INPUT

Java input console is accomplished by reading from System.in. To obtain a character based stream that is attached to the console, we have to wrap System.in in a BufferedReader object, to create a character stream.

The syntax to obtain an object of BufferedReader is:

```
BufferedReaderbr = new BufferedReader(new
InputStreamReader(System.in));
```

Once BufferedReader is obtained, we can use read() method to reach a character or readLine() method to read a string from the console.

Reading Characters from Console

To read a character from a BufferedReader, we would use read( ) method whose syntax is as follows: int read( ) throws IOException

Each time that read( ) is called, it reads a character from the input stream and returns it as an integer value. As you can see, it can throw an IOException. The following program demonstrates read( ) by reading characters from the console until the user types a "q":

Example 11.14: Program to show use of read()

```
// Use a BufferedReader to read characters from the
console.
import java.io.*;
public class BRRead
{ public static void main(String args[]) throws
IOException
{ char c; // Create a BufferedReader using System.in
BufferedReaderbr = new BufferedReader(new
InputStreamReader(System.in)); System.out.println("Enter
characters, 'q' to quit."); // read characters
do { c = (char) br.read(); System.out.println(c); }
while(c != 'q');
} }
```

Reading Strings from Console

To read a string from the keyboard, we have to use readLine( ) that is a member of the BufferedReader class. Its general form is:

```
String readLine( ) throws IOException
```

The following program demonstrates BufferedReader and the readLine( ) method. The program reads and displays lines of text until you enter the word "end":

Example 11.15: Program to show use of readLine()

```
// Read a string from console using a BufferedReader.
import java.io.*;
public class BRReadLines
{ public static void main(String args[]) throws
IOException
{ // Create a BufferedReader using System.in
BufferedReaderbr = new BufferedReader(new
InputStreamReader(System.in));
String str;
System.out.println("Enter lines of text.");
System.out.println("Enter 'end' to quit.");
do { str = br.readLine(); System.out.println(str);
} while(!str.equals("end"));
}
}
```

There are several ways to read data given by the user at the console. They are:

- Scanner class
- BufferedReader class
- Console class

Scanner Class:

This is the most common and preferred method to take input. It is used to read input from the console. However, it was initially brought up for the purpose of parsing strings and primitive data types. With this feature of parsing the tokenized input using regex and a series of methods that can be used to parse primitive data, makes it a popular choice to read input.

Example 11.16: Program to show use of Scanner class

```
import java.util.Scanner;
public class ScannerTest
{
    public static void main(String args[])
    {
```

```
            // Using Scanner for Getting Input using
System.in[ Default Configuration is Console ]
        Scanner in = new Scanner(System.in);
System.out.println("Please enter some string: ");
        String myString = in.nextLine(); //This parses
to string

System.out.println("Please enter some integer: ");
intmyInt = in.nextInt(); //This parses to integer

System.out.println("Please enter some float: ");
        float myFloat = in.nextFloat();  //This parses
to float

System.out.println("*******Validation Part*******");
System.out.println("Your input string: "+myString);
System.out.println("Your input int: "+myInt);
System.out.println("Your input float: "+myFloat);
    }
}
```

Output:

```
Please enter some string: Arunabha Das
Please enter some integer: 60
Please enter some float: 25.87
*******Validation Part*******
Your input string: Arunabha Das
Your input int: 60
Your input float: 25.87
```

BufferedReader Class

This is the classical way to take input, which was part of JDK's first release. Here, we instantiate the BufferedReader class by wrapping an InputStreamReader which wraps the standard console input stream. It buffers the input for efficient reading. It is accompanied by static methods

of Wrapper classes to read corresponding data types.

Program 11.17: Program to show use of BufferedReader class

```
import java.io.*;
public class BufferedReaderTest
{
      public static void main(String args[]) throws
IOException
    {
      // Using Scanner for Getting Input using System.in
BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
System.out.println("Givea string: ");
        String myString = in.readLine(); //This parses
to string

System.out.println("Please enter some integer: ");
intmyInt = Integer.parseInt(in.readLine()); //This parses
to integer

System.out.println("Enter some float value: ");
      float myFloat = Float.parseFloat(in.readLine());
//This parses to float

System.out.println("*******Validation Part*******");
System.out.println("Your input string: "+myString);
System.out.println("Your input int: "+myInt);
System.out.println("Your input float: "+myFloat);
    }
}
```

Output:

```
Give a string: Reading console input
Please enter some integer: 45
Enter some float value: 15.67
```

```
*******Validation Part*******
Your input string: Reading from console
Your input int: 45
Your input float: 15.67
```

Console Class

It is another for reading the user's input from the command line being synchronized. In addition, it has the feature of maintaining secrecy while reading password-like input without echoing the input. The string can also be formatted.

Program 11.18: Program to show use of Console class

```
public class MainConsoleTest
{
    public static void main(String args[])
    {
                            String   myString   =
System.console().readLine("Please enter username: ");
char[] myPassword = System.console().readPassword("Please
enter password: ");

System.out.println("*******Validation Part*******");
System.out.println("Your input username: "+myString);
System.out.print("Your input pasword: ");
for(char x:myPassword){
System.out.print(x);
        }
    }
}
```

Output:

```
Please enter username: Robert Perry
Please enter password: *****
*******Validation Part*******
Your input username: Robert Perry
Your input pasword: 12345
```

11.6 WRITING CONSOLE OUTPUT

Console output is most easily accomplished with print() and println() methods which are defined by the class PrintStream which is the type of the object referenced by System.out. Because PrintStream is an output stream derived from OutputStream, it also implements the low?level method write(). Thus, write() can be used to write to the console. The simplest form of write() defined by PrintStreamis :

void write(intbyteval);

This method writes to the stream the byte specified by byteval. Although byteval is declaredas an integer, only the low?order eight bits are written.

A simple example demonstrating write( ) to output the character "A" followed by a newline to the screen is given below:

```
import java.io.*; // Demonstrate System.out.write().
public class WriteDemo{ public static void main(String args[])
{ int b;
b = 'A';
System.out.write(b);
System.out.write('\n');
 }
}
```

There are two ways to write at the console. Both the ways belong to class PrintStream(Character-based class).

- **print and.println methods:** Here, the methods are defined by class Printstream and is referenced by System.out.
- **write method:** It is also defined by Printstream class. It is not recommended for use.

Let us look at the following example.

Program 11.19: Program to show use of write().

```
public class Main
{
```

```
        public static void main(String args[])
        {
    System.out.println("System.out.print will print the
passed string");
    for(inti=1;i<=5;i++){
    System.out.print(i);
          }

    System.out.println("\nSystem.out.printlnprints the
passed string and end with a new line");
    for(inti=1;i<=5;i++)
    {
    System.out.println(i);
          }
    System.out.println("\nSystem.out.write accepts the
intbyteval as input parameter ");
    for(int x=65;x<=70;x++){
    System.out.write(x);
    System.out.write('\n');
          }
        }
    }
```

Output:

```
    System.out.print will print the passed string.
    12345
    System.out.println prints the passed string and
end with a new line
    1
    2
    3
    4
    5
```

```
System.out.writeaccepts the intbyteval as input
parameter
A
B
C
D
E
F
```

11.7 READING AND WRITING FILES

To read from or write to files, three important streams FileInputStream, ByteArrayInputStream and DataInputStreamare usedfor inputs and three streams FileOutputStream, ByteArrayOutputStream and DataOutputStream are used for outputs.

File Input Stream

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.

A constructor of FileInputStreamtakes a file name as a string to create an input stream object to read the file as shown below.

```
InputStream f = new FileInputStream("C:/java/
hello");
```

Another constructor of File class takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows:

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Once you have InputStream object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

Byte Array Input Stream

The ByteArrayInputStream class allows a buffer in the memory to

be used as an InputStream. The input source is a byte array. There are following forms of constructors to create ByteArrayInputStream objects.

By taking a byte array as the parameter:

```
ByteArrayInputStreambArray = new ByteArrayInputStream(byte [] a);
```

Another form takes an array of bytes, and two integers as arguments, where off is the first byte to be read and len is the number of bytes to be read.

```
ByteArrayInputStreambArray = new ByteArrayInputStream(byte []a, int off, intlen);
```



CHECK YOUR PROGRESS

Q 1. Which of these class contains the methods used to write in a file?

a) FileStream b) FileInputStream c) BufferedOutputStream d) FileBufferStream?

Q 2. Which of these methods are used to read in from file?

a) get() b) read() c) scan() d) readFileInput()

Q 3. Which of these exception is thrown by close() and read() methods?

a) IOException b) FileException c) FileNotFoundException d) FileInputOutputException

Q 4. Which of these classes is used for file handling?

a) InputStream b) PrintStream c) FileWriter d) None of these

Q 5. Which of these classes is not a member of java.io package?

a) File b) Writer c) String d) StringReader

Q 6. Which method is used to write a byte to the current output stream?

a) public void write(int) throws IOException

b) public void write(byte[]) throws IOException

c) public void flush() throws IOException

d) public void close()throws IOException

Q 7. Which class in Java is used to take input from the user?

a) Scanner b) Input c) FileInput

d) None of the above

Q 8. It is important to close the stream _____ after writing to a file is finished.

a) Object b) Reader c) File d) Directory

Q 9. The abstract class _____ defines the functionality that is available to all character input streams.

a) Object b) Reader c) File d) Directory

Q 10. _____ class converts a stream of bytes into a stream of characters.

a) Writer b) Reader c) String d) None of these



11.8 LET US SUM UP

- In Java standard I/O, inputs and outputs are handled by the so-called streams.
- A stream is a sequential and logical representation of one-way flow of data.
- Stream I/O operations involve three steps:
  - Open an input/output stream associated with a physical device (e.g., file, network, console/keyboard), by constructing an appropriate I/O stream instance.
  - Read from the opened input stream until "end-of-stream" encountered, or write to the opened output stream (and optionally flush the buffered output).
  - Close the input/output stream.
- Based on the data type, there are two types of streams: Byte Stream and Character Stream.
- Byte Stream is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.

- Character Stream is mainly involved with character data. A file handling process with a character stream is a process in which an input is processed with the character data.
- Depending on the type of operations, byte streams can be classified into two main groups: InputStream and OutputStream.
- InputStream is used to read data that must be taken as an input from a source array or a file or any peripheral device. For example, FileInputStream, BufferedInputStream, ByteArrayInputStream, etc. are some predefined input stream classes.
- OutputStream is used to write data as outputs into an array or file or any output peripheral device. For example, FileOutputStream, BufferedOutputStream, ByteArrayOutputStream etc. are predefined streams under this group.
- There are 3 standard or default streams System.in, System.out and System.err that Java provides for I/O operations.
- Five operations on a file can be performed- Creating a file, Reading from a file, Writing to a file, Deleting a file and Retrieving file operation.



11.9 FURTHER READING

1) Schildt, H. (2014). Java: The Complete Reference, Oracle Press.
2) Rao, R. N. (2006). Core and Advanced Java Black Book, Dreamtech Press



11.10 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. b

Ans to Q No 2. b

Ans to Q No 3. a

Ans to Q No 4. a

Ans to Q No 5. c

Ans to Q No 6. a

Ans to Q No 7. a

Ans to Q No 8. a

Ans to Q No 9. b

Ans to Q No 10. b



11.11 MODEL QUESTIONS

Q 1. Discuss about basics of I/O in Java.

Q 2. What is stream meant for? What are the streams used for I/O?

Q 3. Explain about three Standard I/O streams.

Q 4. How do you implement Reading Console input and Writing Console output in Java?

Q 5. Write a Java program to create a file "Welcome.txt" and write one line of text into that file.

Q 6. What are the operations be performed on files? Discuss various Predefined streams for file operations in Java.

*** ***** ***

UNIT 12: INTRODUCTION TO APPLETS

UNIT STRUCTURE

12.1 Learning Objectives

12.2 Introduction

12.3 Applet

12.4 Applet and the World Wide Web

12.5 The Applet Class

12.6 Applet and Html

12.7 Life Cycle of an Applet

12.8 Using Window Components

12.9 Event Handling

12.10 Adding Audio and Animation

12.10.1 Adding Audio in Applet

12.10.2 Adding Animation in Applet

12.11 Let Us Sum Up

12.12 Further Reading

12.13 Answer to Check Your Progress

12.14 Model Questions

12.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define Applet
- describe relation between Applet and World Wide Web
- define Applet class and Applet class hierarchy
- define HTML and Applet
- explain the creating steps of Applet
- describe the life cycle of an Applet
- explain the use of Window component in Applet
- describe event handling in Applet
- explain how audio is added in Applet
- explain how animation is added in Applet

12.2 INTRODUCTION

In the earlier unit, we have learnt about the concept of file handling in detail. We have learnt how to read and write in console and tiles in the previous unit.

In this unit, you will be introduced to applets. Topics like applet and world wide web, html, applet class and the life cycle of applet are discussed in detail in this unit. In addition to these, concepts related to window components, event handling and addition of audio and animation to applets are also described in this unit. In the next unit, we will explore the topics related to AWT and Swings.

12.3 APPLET

Applet is a small java program which can be easily passes through the network from one computer to another and run using the Applet viewer or any other web browser that can supports Java. It can do the display graphics, arithmetic operation, play sound, take user input etc. It creates a fully inactive multimedia web application. We can embed applet into web pages into two ways. First is create our own applet and embedded into a web page. Second is download applet from a remote machine and embedded into web pages. First is known as local applet where second one called as remote applet. To locate and load the remote applet we must know the URL of the remote applet.

12.4 APPLET AND WORLD WIDE WEB

The "applet" is represent a small, subordinate, or embeddable application which means it's designed to be run and used within the context of a larger system. Most of the applet programs are embedded within a computer's operating system where operating system manages its native applications such as it starts, stops, suspends, and synchronizes applications; it provides them with certain standard resources; and it protects them from one another by partitioning their environments. An applet is just another type of object to display on web browser; it's embedded into an

HTML page with a special tag. It runs in a Web browser and works at client side. Browsers distinguish the inline items and items hosted via hypertext links i.e. available from external sources such as a viewer or helper application.

The safety for an application is a features of Java which make it possible to develop new kinds of applications thatwere infeasible. A web pages make with applet can contain not only static hypertext information but also full-fledged interactive applications. A Web browser navigate the user to retrieve the data and use software. Initially static information was related with portable software for interpreting and using the information which was replace a fully functional spreadsheet application embedded within it that allows users to view and manipulate the information.

Important points related to Applet

- All applets are sub-classes of java.applet.Applet class.
- Applets are not stand-alone programs. Instead, so they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
- In general, execution of an applet does not begin at main() method.
- Applet cannot perform read or write operation on a file in the local computer.
- Applet cannot communicate with other server on the network
- Applet cannot run any program from the local computer.
- Applets are restricted from using libraries from other languages.
- Output of an applet window is not performed by System.out.println().it is handled with various AWT methods, such as drawString().

Why Applet is needed?

- When something dynamic to be included in the display of a web page.
- When we require some flash output.
- When we want to create a program and make it available on the internet for us by the others on their computer.

Advantage of Applet:

There are many advantages of the applet. They are as follows:

- It works on the client side so less response time.

- It secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os, etc.

Disadvantage of Applet:

- To execute the applet, plugging is required in the client browser.

12.5 APPLET CLASS

To create an applet in Java, the user can use java. applet.Applet class. There are four Methods used in the Life Cycle of the Java Applet. JDK provides an applet viewer utility to execute the applet. A Java Applet class, a part of java. applet package is responsible to create java Applet, The Applet class shows a standard interface between applets and their environment. The subclass of the Applet class is called an applet that is embedded in a Web page or viewed by the Java Applet Viewer. The Java applet class have some useful methods to give you complete control to run an Applet like initializing and destroying an applet, It also provides the methods to load and display colorful images, and play audio and video clip and Cinematic Videos on the Web.

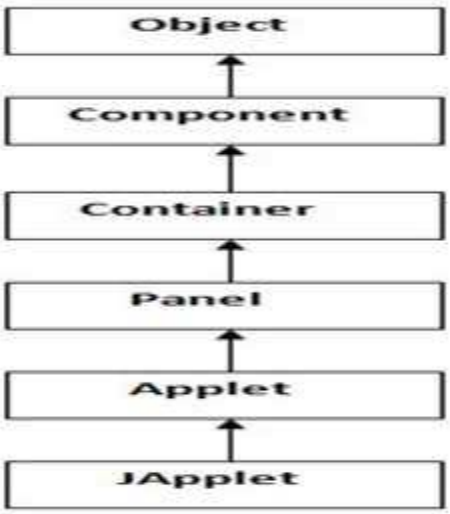


Figure 12.1: Hierarchy of applet class

12.6 APPLET AND HTML

Applets are contained in web documents whose structure is defined in HTML. In HTML, tags like head, title, body, etc. are used to define different parts like heading, the title of the page, and what content would like to display. To display applet programming, an applet tag is used with attribute CODEBASE(optional), code, ALT, width, height, alignment, and so on.

Table 12.1: Table for the attribute of the Applet tag

Attribute name	Value	Description
code	URL	It specifies the URL of the Java applet class file.
width	pixels	It specifies the display width of the applet panel.
height	pixels	It specifies the display height of the applet panel
align	Left, right, top, bottom, middle	It specifies the position of the applet application relative to the surrounding content.
alt	text	It is used to display alternative text in case the browser does not support Java.
archive	URL	This specifies the archived or compressed version of an applet application.
object	name	It specifies the URL or reference to a serialized representation of an applet.
codebase	URL	It specifies the exact or relative URL of the applets .class file specified in the code attribute.
space	pixels	It specifies the horizontal space around the applet.
space	pixels	It specifies the vertical space around the applet.
name	name	It specifies the name of the applet

Steps to developing an applet

- Build an applet code(.java file).
- Creating an executable applet (.class file)
- Designing web page using HTML tags

- Preparing <APPLET>tag.
- Incorporating <APPLET> tag into the web page.
- Creating HTML file.
- Testing the applet code.

How to run an applet?

There are two ways to run the applet

- By Web browser: open the HTML file using that browser
- By appletviewer tool: This is the easiest way to run an applet.

To execute a file with an applet viewer use code

appletviewer filename.html

Features of Applets over HTML

- Displaying dynamic web pages of a web application.
- Playing sound files.
- Displaying documents
- Playing animations

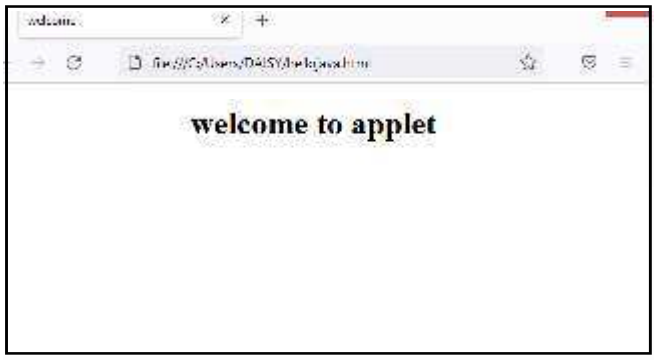
Program12.1: Program to display a line in the applet application

a. Code to create a java file

```
import java.awt.*;
import java.applet.*;
public class hellojava extends Applet
{ public void paint(Graphics g)
{g.drawString("hello java", 10,1000);
}
}
<html>
<head><title>
welcome</title></head>
<body>
<center>
<h1>welcome to applet</h1>
</center>
<br>
<center>
```

```
<applet
   code=hellojava.class
   width=400
   height=200>
</applet>  </center>
</body>    </html>
```

Output:





CHECK YOUR PROGRESS

Q 1. What is Applet?

Q 2. Write two features of Applet.

Q 3. Why the applet is necessary?

Q 4. Write name of two attributes of applet tag.

Q 5. How can you run an applet?

12.6 THE LIFE CYCLE OF AN APPLET

Java applet inherits the set of default behavior from the Applet class. When an applet is loaded it goes through the different applet states like born or initialize state, running state, idle or stopped state, and dead or destroyed state. Java plug-in software is responsible for managing the life cycle of an applet.

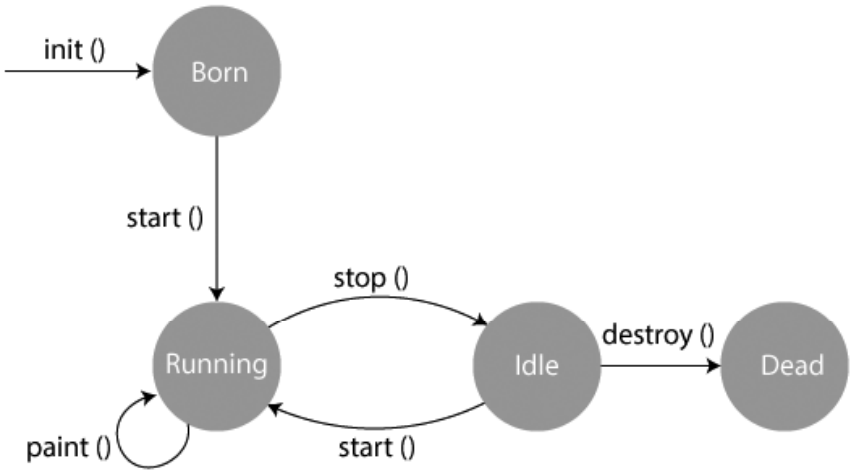


Figure 12.2: Life cycle of an Applet

Initialization state:

It goes through the following process

- The first stage of an applet. So create objects needed by the applet and it invokes only once.
- Set up initial values
- Load images or fonts
- Set colors

Running State:

- It executes when the system calls the start() method of the Applet class
- Execute after the init() method.
- Sometimes start() is occur if the applet is in a stopped / idle state.
- It can be invoked more than one time.

Idle or Stopped state

- Execute when stop from running
- Automatically execute when leaving the page containing the currently running applet.
- We can call stop() explicitly.
- Terminate the thread on the execution of stop() if the applet runs on a thread.

Dead State

- Used to destroy the applet.

- Automatically invoked when quitting the browser.
- Occur only once time
- Clean up the resources.

Display state

- Execute when the applet has to perform some output operation.
- Execute immediately after the applet enters into running state.
- Paint() accomplished this task.

12.7 USING WINDOW COMPONENTS

In java, the AWT class gives the facility to add different types of window components in the applet. AWT has a rich set of components that can be used in the window. In the applet, we can add graphical buttons, labels, text fields, etc.

Program 12.2: Program on user interactive applet

```
import java.awt.*;
import java.applet.*;
public class AWTapplet extends Applet
{     Button b1;
TextFieldtx;
Label l1;
      public void init()
{
            b1=new Button("B1");
            tx=new TextField("This is textfield");
            l1=new Label("Button not pressed");
            add(b1);
            add(tx);
            add(l1);
}
}
```

12.8 EVENT HANDLING

Changing the state of a window object is known as an event. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler that is executed when an event occurs. To take the input from the button and change the content of the label as the output of actionPerformed(). This method includes a class that implements ActionListenerInterace.

Program 12.3: Program context input by the user in an applet program

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class Textinput extends Applet implements
ActionListener
{
TextField name, age;
      Public init()
      {
setSize(300,200);
            Label l1=new Label("Name");
            Label l2=new Label("Age");
            name=new TextField(25);
            age=new TextField(4);
            add(l1);
            add(l2);
            add(name);
            add(age);
            name.addActionListener(this);
            age.addActionListener(this);
}
Public void actionPerformed(ActionEvent ac)
{
```

```
repaint();
}
Public void paint( Graphics g)
{
g.drawString("Name: "+name.getText(),10,100);
g.drawString("Age: "+age.getText(),10,130);
}
}
```

12.9 ADDING AUDIO AND ANIMATION

In this section, we will learn how to add audio to an applet. We will also learn how to add animation to an applet in this section.

12.9.1 Adding Audio in Applet

The AudioClip class is used to load and play Audio files. To load a sound file the getAudioClip () Method is used. The general format of the getAudioClip () method is:

```
AudioClipgetAudioClip (URL pathname, String filename)
AudioClipgetAudioClip (URL pathname)
```

where,

pathname defines the address of the sound file,

the filename is the name of the sound file.

When the image file and the source file are in the same directory, getCodeBase () method is used as the first parameter to the method. Some of the methods from AudioClip class are void play(), void loop(), and void stop().

Program 12.4: Program to play audio clip in Applet

```
import java.applet.*;
import java.awt.*;
public class audioex extends Applet
{
```

```
private AudioClipmysnd;
      public void init()
{
       mysnd=getAudioClip(getCodeBase(),"Audio
filename.mp3");
     }
      public void start()
     {
mysnd.loop();
      }
     public void stop()
    {
mysnd.stop();
     }
}

HTML code for sound example
<HTML>
        <HEAD>
        </HEAD>
<BODY>
        <CENTER>
         <APPLET  CODE="audioex.class"  WIDTH=200
HEIGHT=130>
       </APPLET>
        </CENTER>
</BODY>
</HTML>
```

12.9.2 Adding Animation in Applet

Java provides support two common image formats: GIF and JPEG. An image of any one of these formats can be loaded by using URL,

or a filename. To get object of image class use a method getImage() with following format:

public Image getImage(URL url. String image);

Some of the other methods related to image or animation are:

- public URL getCodeBased() returns the base URL.
- public URL getDocumentBase() returns the URL of the document in which the Applet is embedded.

Program 12.5: Program to add animation in applet

```
import java.awt.*;
import java.applet.*;
public class AnimationEx extends Applet
{
      Image pic;
      Public void init()
{
      pic=getImage(getDocumentBase(),
"images.jpeg");
}
      public void paint(Graphics g)
{
      for (int i = 50; i < 600; i++)
{
      g.drawImage(pic, i, 30, this);
            try
{
      thread.sleep(400);
                        } catch(Exception e)  {}
                  }
            }
      }
```



CHECK YOUR PROGRESS

Q 6. Who is responsible to manage the life cycle of an applet?

Q 7. Which method(s) is/are responsible for display state?

Q 8. What are different state of an applet?

Q 9. What happens when an applet is loaded?



12.10 LET US SUM UP

- Applet programming is important feature of Java programming.
- It is a special kind of programme that is embedded in HTML web page.
- There are two types of applet namely local and remote.
- The HTML Applet tag consists of Codebase, code, align, ALT, height, width etc. attributes.
- Applet programming is related graphics application.
- Life cycle of an applet goes through the five different state.
- AWT components are applicable on applet.
- These components can be activate implementing ActionListener interface.
- Animation and Audio can run using applet.
- The applet class contain 23 methods that are used to get information about the applet.



12.11 FURTHER READING

1) Schieldt, H. (2010), The Complete Reference Java(7th), Tata McGraw Hill
2) Seth, A. &Juneja, B. L. (2017), Java One Step Ahead(1st), Oxford University Press.
3) Balagurusamy, E.(2019), Programming with Java(6th), Tata McGraw Hill.
4) Malhotra, S. & Choudhury, S. (2018), Programming with Java(2nd), Oxford University Press

12.12 ANSWERS TO THE CHECK YOUR PROGRESS

Ans to Q No 1. An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side.

Ans to Q No 2. Two features of applet are

- All applets are sub-classes of java.applet.Applet class.
- In general, execution of an applet does not begin at main() method.

Ans to Q No 3. The applet is necessary:-

- When something dynamic to be included in the display of a web page.
- When we require some flash output.
- When we want to create a program and make it available on the internet for us by the others on their computer.

Ans to Q No 4. Code and height.

Ans to Q No 5. Using Web browser or by using appletviewer.

Ans to Q No 6. Java plug-in software.

Ans to Q No 7. drawstring(), or paint()

Ans to Q No 8. Initialization, running, stop and dead state.

Ans to Q No 9. First of all, an instance of the applets controlling class is created. Then, the applet initializes itself and finally, it starts running

12.13 MODEL QUESTIONS

Q 1. What is Applet? What are features of applet differentiate it from other application?

Q 2. Write a applet program to take different values from user input and display addition of the no.

Q 3. Describe the different state of applet life cycle.

Q 4. Write about the hierarchy of applet class.

Q 5. What are the functions of different attribute used in Applet tag?

Q 6. Write the steps to create an applet.

Q 7. Write format of paint() and drawString() with example.

Q 8. Write an applet program to display calculator functions.

Q 9. Write about the procedure to add window component and event handling.

Q 10. Write an applet program to display animation with sound.

*** ***** ***

UNIT 13: AWT AND SWINGS

UNIT STRUCTURE

13.1 Learning Objectives

13.2 Introduction

13.3 AWT

13.3.1 AWT Basics

13.4 AWT Components

13.5 Event Handling with AWT

13.6 Application and Menus

13.7 Introduction to Swings

13.8 Swing Components

13.9 Event Handling with Swing

13.10 Display Text and Image in a Window

13.11 Layout Manager

13.12 Let Us Sum Up

13.13 Further Reading

13.14 Answer to Check Your Progress

13.15 Model Questions

13.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- define AWT and describe its components
- describe the hierarchy of components
- define Container, Button, Label, Checkbox
- describe event handling with AWT
- explain how to create Window, Menus and MenuBar
- define Swing and describe its components
- describe event handling in Swing
- describe how to display image and Text
- describe the different Layout Manager

13.2 INTRODUCTION

In the earlier unit, we had been introduced to applets. Topics like applet class, the life cycle of applet concepts related to window components, event handling, and the addition of audio and animation to applets were described in the previous unit.

In this unit, you will be introduced to AWT and Swings. Topics like AWT basics, AWT components, event handling in AWT, and application and Menus are discussed in detail in this unit. In addition to these, concepts related to Swings, swing components, event handling in Swings, etc. are also described in this unit. In the next unit, we will explore the topics related to JDBC.

13.3 AWT

The Abstract Window Toolkit (AWT) is a large package in Java that supports the creation and management of windows as well as graphics. Different shapes like points, lines, polygons arcs may be drawn in the window. It also manages the colors, different alignment images, text, etc. The window may be created through the standalone program or as an applet. All the objects used in AWTprovided by java.awt package. It provides an Interactive GUI to the user.

Swing in Java is a lightweight GUI toolkit that has a wide variety of widgets for building optimized window-based applications. It is a part of the JFC(Java Foundation Classes).Like AWT it has different components through which user can input their data. All the components are provided by java.awt and javax.swing

13.3.1 AWT BASICS

AWT stands for Abstract Window Toolkit which is a Java Application programming interface (API) to create a Graphical User Interface (GUI). It allows Java programmers to develop window-based applications.

AWT provides components like buttons, labels, checkboxes, etc. which can be used as objects inside a Java Program. AWT components are platform-dependent which means that the component's view can be changed according to the operating system's view and they use the resources of the operating system. The classes from Java.awt package provide the various AWT components.

13.4 AWT COMPONENTS

All the elements like the button, text fields, scroll bars, etc. are called components. To place every component in a particular position on a screen, we need to add them to a container.

Hierarchy of Java AWT Classes

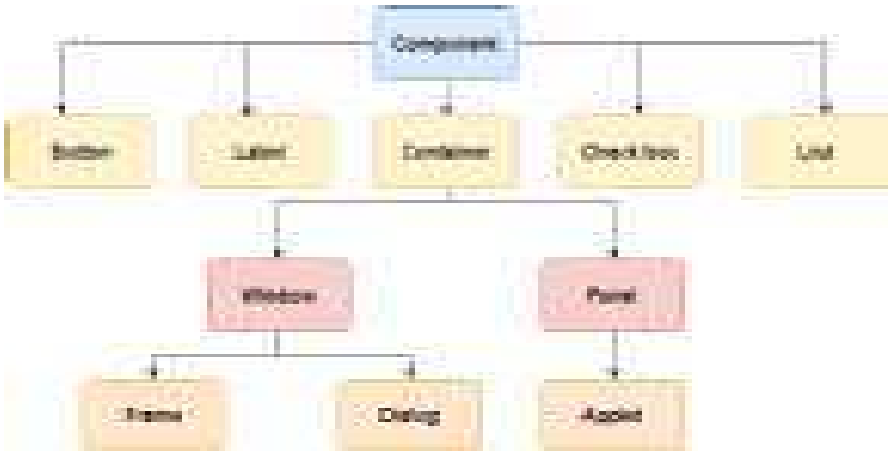


Figure 13.1: Hierarchy of Java AWT Classes

Container

The Container is a component in AWT that can contain other components like buttons, text fields, labels, etc. It is a screen and encapsulates all the attributes of the visual component. The classes that extend the container class are known as containers such as Frame, Dialog, and Panel. Thus it contains and controls the layout of components.

There are three types of containers in Java AWT:

1. Window
2. Panel

3. Frame
4. Dialog

1. **Window:** The window is the container that has no borders and menu bars. You must use a frame, dialog, or another window for creating a window. You need to create an instance of the Window class to create this container
2. **Frame:** Frame is a subclass of Window that can contain a title, border, and menu bars. Resizing canvas is one of the features and is the widely used container for developing AWT applications. It is capable to hold various components such as buttons, text fields, scrollbars, etc. it can create a Java AWT Frame in two ways:
   i. By Instantiating Frame class
   ii. By extending the Frame class
3. **Dialog:** The dialog class is also a subclass of Window and comes with the border and title. To exist an instance of the Dialog class always needs an associated Frame class instance.
4. **Panel:** The Panel is the container that doesn't contain a title bar, border, or menu bar. It is a generic container for holding the GUI components. It can have other components like buttons, text fields, etc. An instance of the Panel class creates a container, in which we can add components.

Canvas: It is not a part frame window but it is valuable. A Canvas class provides a rectangular area where the user can draw in an application or receive inputs created by the user.

Other Components:

- **Button:** java.awt.Button class is used to create a labeled button. By clicking this component you can trigger a certain programmed action.
- **Text Field:** It is a component of java.awt.TextField class which creates a single-line text box for users to enter texts.
- **Label:** It provides a descriptive text string that is visible on GUI. An AWT Label object is a component for placing text in a container.
- **Choice:** Choice class is used to represent a pop-up menu of choices. The selected choice is shown at the top of the given menu.

- **Scroll Bar:** The Scrollbar class object is used to add horizontal and vertical scrollbars in the GUI. It enables a user to see the invisible number of rows and columns.
- **List:** The object of the List class represents a list of text items. Using the list class a user can choose either one item or multiple items.
- **CheckBox:** The Checkbox is a class that is a graphical component that is used to create a checkbox. It has two state options; true and false.

13.5 EVENT HANDLING WITH AWT

Changing the state of an object is known as an event. Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler that is executed when an event occurs. For example, when a user clicks on the button, an event is created and sent to one or more Listeners that handle the event.

Table 13.1: Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListenerand MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Window Display System

The window is a basic display area where we can display other components such as buttons, checkboxes, labels, etc. However, this window cannot be added to another AWT window. The layout of the components in the window is managed by the layout manager set by the programmer.

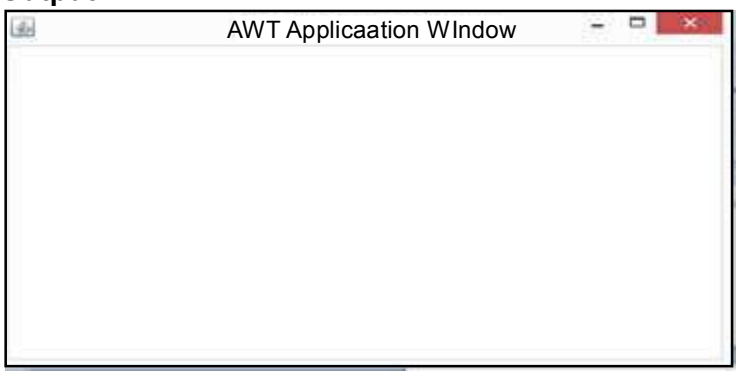
Program 13.1: Program to create a blank window with a closing event

```
import java.awt.*;
import java.awt.event.*;

public class AwtWindow extends Frame
{
public static void main(String args[])
{
AwtWindow win=new AwtWindow();
//create an object of class
}
public AwtWindow()          // construct a method
{
super("AWT Application Window");
setSize(500,300);           //set size of window
addWindowListener(newAwtWindow.WindowEventHandler());

setVisible(true);
}
class WindowEventHandler extends WindowAdapter{
public void window closing(WindowEvent we)
{
System.exit(0);
}
}
}
```

Output:



Adding Components to the Window:

Generally, to add the components to the window using the method add(). The procedure is as follows:

1. Create the component to be added as an object of the relevant class.

2. Use method adds () to add already created components to the window.

If a component is not further required on the window it can be removed by using the method remove().

Steps to perform Event Handling

The following steps are required to perform event handling:

- Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}

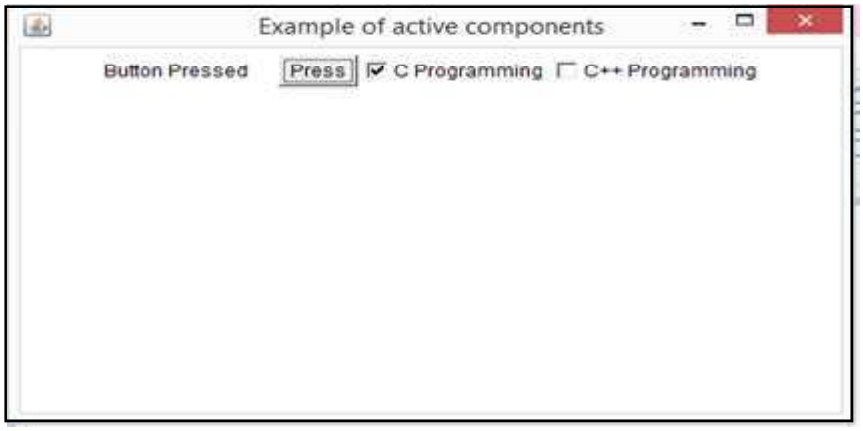
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
    - public void addItemListener(ItemListener a){}

Program 13.2: Program to add and work with label, button, and checkbox

```
      import java. awt.*;
      import java.awt.event.*;
      public class input work extends Frame implements
ActionListener
      {
      public static void main(String args[])
      {
      inputWork iw=new inputWork();
      }
      Label lbl=new Label();     //create a label
      Button b1=new Button("Press");
      //create a button
      public inputWork()
      {
      super("Example of active components");
      // setting window title
      setSize(500,300);
      setLayout(new FlowLayout(FlowLayout.CENTER));
//set window layout
      lbl.setText("Press Button.....");
      //define the label
      add(lbl);                  //add label
      add (b1);                  //add button
      b1.addActionListener(this);//activate button
      Checkbox c1=new Checkbox("C Programming");
      //create checkbox
```

```
Checkbox c2=new Checkbox("C++ Programming");
add(c1);                    //add checkbox
add(c2);
addWindowListener(new WindowAdapter()
//closing window
{
public void window closing(WindowEvent e)
{System.exit(0);}
}
);
setVisible(true);
}
public void actionPerformed(ActionEvent ae)
{
lbl.setText("Button Pressed");
}
}
```

Output:



13.6 APPLICATION AND MENU

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.The object of Menu class is a pull down menu component which

is displayed on the menu bar. It inherits the MenuItem class.Java provide several classes to create menus, adding menuitem to menubar and submenus but all these classes are related to MenuComponent.

The process to create menus involve the following steps:

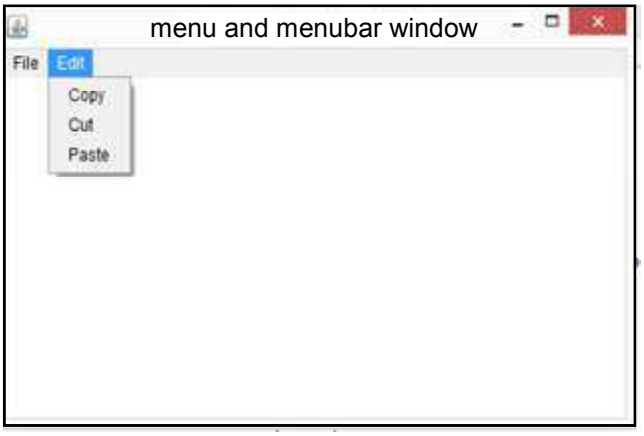
- Create a menubar
- Create menu
- Create and add menu items to menus
- Add menus to the menubar
- Set the menubar

Program 13.3: Program to create Menubar and add a menu item

```
import java.awt.*;
import java.awt.event.*;
public class mymenus extends Frame
{
public static void main(String args[]){
mymenus menub=new mymenus();
}
MenuBar mb=new MenuBar();
//create menubar object
public mymenus()
{
super("menu and menubar window");
setSize(500,300);
setLayout(new FlowLayout());
setMenuBar(mb);
Menu fmenu=new Menu("File");
//create menu
MenuItem fnew=new MenuItem("New");
//create menuitems
MenuItem fopen=new MenuItem("Open");
MenuItem fsave=new MenuItem("Save");
fmenu.add(fnew);
```

```
//add menu item to menu
fmenu.add(fopen);
fmenu.add(fsave);
Menu emenu=new Menu("Edit");
MenuItem ecopy=new MenuItem("Copy");
MenuItem ecut=new MenuItem("Cut");
MenuItem epaste=new MenuItem("Paste");
emenu.add(ecopy);
emenu.add(ecut);
emenu.add(epaste);
mb.add(fmenu);   //add menu to menubar
mb.add(emenu);
addWindowListener(new WindowAdapter()
//closing window
{
public void windowClosing(WindowEvent e){
System.exit(0);
}
});
setVisible(true);
}
}
```

Output:





CHECK YOUR PROGRESS

Q 1. What is Component?

Q 2. What is Window?

Q 3. Describe the steps to add component.

Q 4. What is the method to register a list in window?

Q 5. How can you create a menu in java AWT?

13.7 INTRODUCTION TO SWINGS

Java Swing is a part of Java Foundation Classes (JFC) which was considered for enabling java applications for large-scale enterprise development.Java Swing provides the graphical user interface (GUI) for Java programs. Java Swing is also known as the Java GUI widget toolkit.Java Swing or Swing is based on earlier APIs called Abstract Windows Toolkit (AWT). But it provides richer and more sophisticated GUI components than AWT. The GUI components are ranging from a simple label to a complex tree and table. Swing is a platform-independent and enhanced MVC (Model -View - Controller) framework for Java applications.

Features of Swing

Some of the common features of Swing are:

- Swing has a Pluggable look and feel. So users can make their own choice to choose which look and feel is the best for them instantly without restarting the application.
- Swing components are lightweight.
- Swing has a number of built-in controls: tree, tabbed panes, sliders, toolbars, tables, etc.
- Users can customize their own GUI application using a border, text alignment, adding images, etc.
- The swing is not thread-safe.
- All drawings can be done using paintComponent rather than paint.
- All Swing components begin with 'J'

13.8 SWING COMPONENTS

Swing components are the basic building blocks of an application. We know that Swing is a GUI widget toolkit for Java. Every application has some basic interactive interface for the user. For example, a button, checkbox, radio button, text field, etc. These together form the components of the Swing. Generally, swing components are derived from JComponent class. It inherits the AWT classes container and component. All the swing components are represented by the classes defined within the packages java. swing. The following table shows the class name of swing components.

Table 13.2: Table for the class name of Swing Components

JApplet	JButton	JCheckBox	JCheckBoxMenuItem
JColorChooser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPane	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayeredPane
JList	JMenu	JMenuBar	JMenuItem
JOptionPane	JPanel	JPasswordField	JPopupMenu
JProgressBar	JRadioButton	JRadioButtonMenuItem	frontpage
JScrollBar	JscrollPane	Jseparator	JSlider
JSpinner	JSplitPane	JTabbedPane	JTable
JTextArea	JTextField	JTextPane	JToggleButton
toolbar	tooltip	JTree	Jviewport
window			

Swing Containers

Swing defines two types of containers. The first are top-level containers: JApplet, JFrame, JWindow, and JDialog. These are inherent in the AWT Component and Container instead of JComponent. Top-level containers are heavyweight. The second type of container supported by Swing is a lightweight container like JPanel. The lightweight containers inherit JComponent.

13.9 EVENT HANDLING IN SWING

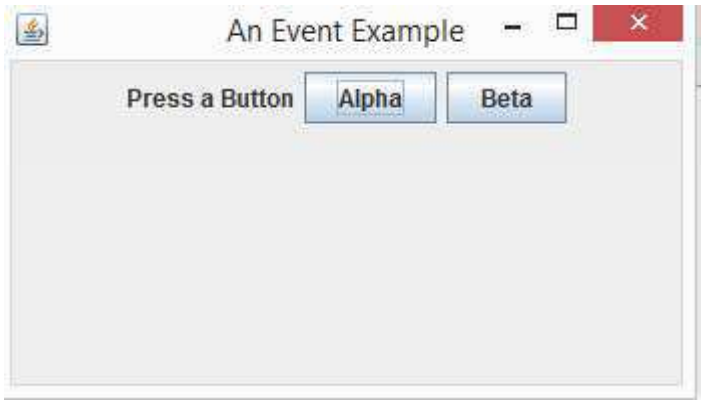
As event handling is a large part of the Swing Application. An event may be generated by an interaction by the user which has to be handled. The event-handling mechanism used by Swing is the same as that used by AWT. This approach is called the delegation event model and those events are packaged in java.awt.event. The events which are only specific to Swing are packaged in javax.swing.event.

Program 13.4: Program to activate components in Swing

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class DemoEvent
{JLabel jlbl;
DemoEvent(){
JFrame jfrm=new JFrame("a=Event Example");
//create a new frame
jfrm.setLayout(new FlowLayout());
//specify the layout of frame with initial size
jfrm.setSize(350,200);
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//terminate the program when user close the
application
JButton jbalpha=new JButton("Alpha");
JButton jbbeta=new JButton("Beta");
// create two buttons
jbalpha.addActionListener(new ActionListener(){
      // activate the button Alpha
public void actionPerformed(ActionEvent ae){
jlbl.setText("Pressed Alpha");
}
});
jbbeta.addActionListener(new ActionListener(){
```

```
                    // activate the button Beta
        public void actionPerformed(ActionEvent ae){
        jlbl.setText("Pressed Beta");
        }
        });
        jlbl=new JLabel("Press a Button");
        //create a text based label
        jfrm.add(jlbl);
// add label to the content pane
        jfrm.setVisible(true);
        // display the frame
        jfrm.add(jbalpha);
        jfrm.add(jbbeta);
// add buttons to the content pane
        }
        public static void main(String args[]){
        SwingUtilities.invokeLater(new Runnable(){
        // create the frame on the event dispatcing thread
        public void run() {
        new DemoEvent();
        }
        });
        }
        }
```

Output:



13.10 DISPLAY TEXT AND IMAGE IN A WINDOW

Swing can display an image using JLabel class. JLabel constructor commonly takes two arguments and it describes as follows:

```
JLabel(String str, int align)
Or
JLabel(Icon icon, int align)
```

The first construct a label with text and the second constructor set a label with the image which defines an interface in Swing. The second argument 'align' is optional.

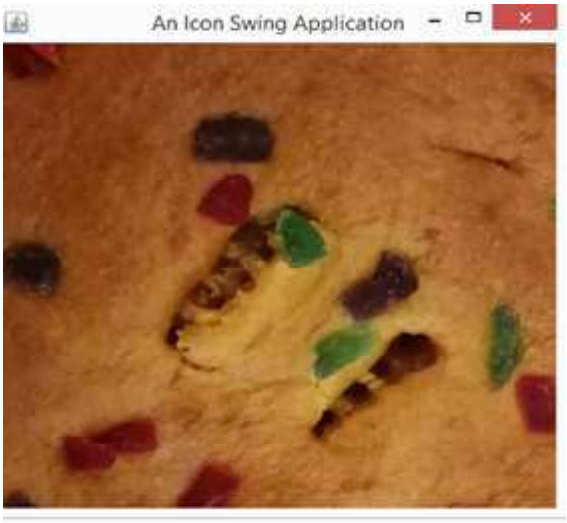
Sometimes we define the JLabel constructor as:

```
JLabel(String str, Icon icon, int align)
```

Program 13.5: Program to display images using Swing

```
import javax.swing.*;
class Appimage extends JFrame{
public static void main(String args[]){
new Appimage();
}
public Appimage()
{setTitle("An Icon Swing Application");
setSize(500,500);
Icon imgIcon=new ImageIcon("Cake.jpg");
//the image
JLabel lbl=new JLabel("Swing Icon", imgIcon,
SwingConstants.CENTER);
getContentPane().add(lbl);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}
}
```

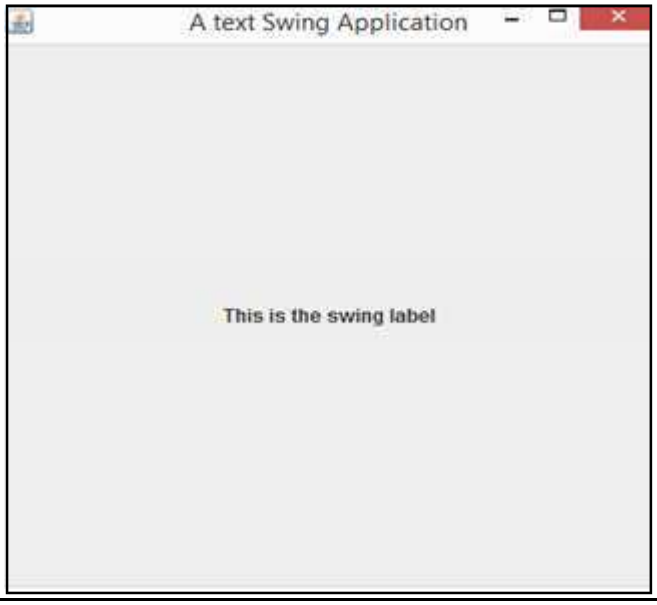
Output



Program 13.6: Program to display Text using Swing

```
import javax.swing.*;
class Appimage extends JFrame{
public static void main(String args[]){
new Appimage();
}
public Appimage()
{setTitle("A text Swing Application");
setSize(400,400);
JLabel lbl=new JLabel("This is the swing label",
SwingConstants.CENTER);
getContentPane().add(lbl);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}
}
```

Output



13.11 LAYOUT MANAGER

Layout Manager in Java swing is used to position all its components, with setting propertiessuch as the size, the shape, and the arrangement. Different layout managers have different settings on their components.

The following are some of layout managers discussed here:

- FlowLayout
- BorderLayout
- GridLayout
- GridBagLayout

FlowLayout

The FlowLayout arranges all the components either from left to right or from right to left. It represents the directional flow. Normally all components are set to one row, if all components cannot be fit into one row, it will start a new row for the rest of the components.

There have three options that could be used to construct FlowLayout:

- **FlowLayout():** construct a new FlowLayout object with center alignment and horizontal and vertical gap of 5 pixels(default).
- **FlowLayout(int align):** construct a similar object with different settings on alignment

- **FlowLayout(int align, int hgap, int vgap):** construct a similar object with different settings on alignment and gaps between components.

BorderLayout

A BorderLayout lays out a container, arranging its components to fit into five different regions: NORTH, SOUTH, EAST, WEST, and CENTER. One component is arranged in one region.

For BorderLayout, it can be constructed like below:

- BorderLayout(): construct a border layout with no gaps between components.
- BorderLayout(int hgap, int vgap): construct a border layout with specified gaps between components.

GridLayout

The GridLayout manager arranged the components in a rectangle grid, which has been divided into equal-sized rectangles and one component is placed in each rectangle. It can construct with the following methods:

- GridLayout(): construct a grid layout with one column per component in a single row.
- GridLayout(int row, int col): construct a grid layout with specified numbers of rows and columns.
- GridLayout(int row, int col, int hgap, int vgap): construct a grid layout with specified rows, columns and gaps between components.

GridbagLayout

GridbagLayout is more flexible than the other layout manager, which allows the components to be vertical, horizontal arrangement without specifying the components to be the same size. Each GridLayout object holds a dynamic rectangular grid of cells. Each GridLayout component is associated with an instance of GridBagConstarintand it decides where the component to be displayed and how the component should be positioned.



CHECK YOUR PROGRESS

Q 6. What is Swing?

Q 7. Which feature give the facility to user to choose their own look in program?

Q 8. What is event handling?

Q 9. What class can be used to display an image in program?

Q 10. What are the different layout available in Swing?



13.12 LET US SUM UP

- The AWT is one of largest package and it supports to create and manage the window and graphics
- The component class is at the top of the hierarchy and it has hundreds of methods.
- Container is a subclass of component class.
- Window class is subclass of container and super class of frame.
- The component like button, label etc. can be add with add().
- If the component is not required then we can remove it also.
- Swing is another package in java which provide same structure like AWT.
- Difference between AWT and Swing is that Swing component are lightweight and AWT components are heavyweight.
- Swing also import AWT package for its heavyweight component.
- Swing also display image, text, label etc. in window.
- Swing components are starts with J.
- AWT and Swing have different Layout managers.



13.13 FURTHER READING

1) Schieldt, H. (2010), The Complete Reference Java(7th), Tata McGraw Hill

2) Seth, A. & Juneja, B. L. (2017), Java One Step Ahead(1st), Oxford University Press

13.14 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. All the elements like the button, text fields, scroll bars, etc. are called components.

Ans to Q No 2. The window is the container that has no borders and menu bars. You must use frame, dialog or another window for creating a window.

Ans to Q No 3. The steps are:

a) Create the component to be added as object of relevant class.

b) Use method add() to add already created component to window.

Ans to Q No 4. Methods:

- public void addActionListener(ActionListener a){}
- public void addItemListener(ItemListener a){}

Ans to Q No 5. The process to create menus involve the following steps:

a. create a menubar

b. create menu

c. create and add menu items to menus

d. add menus to the menu bar

e. set the menubar

Ans to Q No 6. Java Swing is a lightweight Java graphical user interface (GUI) widget toolkit that includes a rich set of widgets.

Ans to Q No 7. The feature that gives the facility to users to choose their look in the program is a pluggable look and feel.

Ans to Q No 8. An event in Java is an object that is created when something changes within a graphical user interface.

Ans to Q No 9. The class that can be used to display an image in a program isJLabel class in Swing.

Ans to Q No 10. The different layouts available in Swing are: FlowLayout, BorderLayout, GridLayout, GridBagLayout

13.15 MODEL QUESTIONS

Q 1. What is AWT? How can it help to create user-friendly application?

Q 2. Explain the hierarchy of AWT Components.

Q 3. Write a program to add four labels and buttons with Center alignment.

Q 4. What is event handling in AWT?

Q 5. What is Swing? Describe its component.

Q 6. Write about the difference between AWT and Swing.

Q 7. Write a program to illustrate a selection of items from the list

Q 8. Describe the different layouts in java swing.

Q 9. What are the constructors of GridLayout and GridBagLayout?

Q 10. Write about the features of Swing in Java. What is a pluggable look and feel?

*** ***** ***

UNIT 14: INTRODUCTION TO JDBC

UNIT STRUCTURE

14.1 Learning Objectives

14.2 Introduction

14.3 JDBC

14.3.1 Basic Steps to JDBC

14.4 JDBC API

14.5 JDBC Drivers

14.6 Connection Management

14.7 JDBC Design Considerations

14.8 Two Tier and Three Tier Client Server Model

14.9 ResultSet

14.10 Prepared Statement and Callable Statement

14.10.1 Prepared Statement

14.10.2 Callable Statement

14.11 ResultSet Meta Data Object

14.12 Let Us Sum Up

14.13 Further Reading

14.14 Answer to Check Your Progress

14.15 Model Questions

14.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe JDBC
- explain the basic steps in JDBC
- describe JDBC API and JDBC drivers
- describe the steps for connections in JDBC
- describe the design of JDBC
- explain the structure of JDBC
- describe Two-tier and Three-tier architecture
- define ResultSet
- describe ResultSetMetaData,

- describe PreparedStatement Interface and CallableStatement Interface

14.2 INTRODUCTION

In the earlier unit, we had been introduced to AWT and Swings. Topics like AWT basics, AWT components, event handling in AWT, and application and Menus were discussed in the previous unit. In addition to these, concepts related to Swings, swing components, event handling in Swings, etc. were also described in the previous unit.

In this unit, you will be introduced to JDBC. Topics like JDBC API and JDBC drivers along with the design and structure of JDBC. In addition to these, concepts related to two-tier and three-tier architecture and ResultSet are also described in this unit. In the next unit, we will explore topics related to database programming using Java.

14.3 JAVA DATABASE CONNECTIVITY (JDBC)

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases. It gives the advantage of connecting the database to java API. All the classes of JDBC The JDBC classes are contained in java.sql and javax.sql packages. JDBC has its API(JDBC API) that uses JDBC drivers. To create the connectivity JDBC follows some steps from loading and registering the driver to closing the connection. JDBC uses four different drivers to connect different types of the data source. It Follows the two-tier and three-tier architecture. A resultset is the return data of the SQL statements that read data from a database query. Users can navigate and update rows and columns in the resultset. JDBC can prepare PreparedStatement and CallableStatement.

14.3.1 Basic Steps to JDBC

In the Java programming language, Java Database Connectivity (JDBC) is an application interface (API) that defines how a client

may access any kind of tabular data, especially a relational database. It gives the service of a middle-layer interface between java applications and databases. The classes and interface of JDBC handle the request made by users to the specified database.

The JDBC classes are contained in java.sql and javax.sql packages.

Why Use JDBC?

Initially, ODBC API was the database API to connect and execute the query with the database. But, the ODBC driver of ODBC API is written in C language. So it is platform dependant and unsecured to overcome this limitation JDBC is introduced against ODBC. It has its API(JDBC API) that uses JDBC drivers (in Java language). JDBC uses its java Native calls.

Fundamental Steps

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

- Import JDBC packages.
- Load and register the JDBC driver.
- Open a connection to the database.
- Create a statement object to perform a query.
- Execute the statement object and return a query resultset.
- Process the resultset.
- Close the resultset and statement objects.
- Close the connection.

14.4 JDBC API

The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, users can virtually access any data source, from relational databases to flat file spreadsheets. JDBC technology provides a common base on which tools and alternate interfaces can be built.

The JDBC API is comprised of two packages:

- java.sql

- javax.sql

To use the JDBC API with a particular database management system, to mediate between JDBC technology and the database, need a JDBC technology-based driver. A driver might be written purely in the Java programming language or a mixture of the Java programming language and Java Native Interface (JNI) native methods. To obtain a JDBC driver for a particular database management system

14.5 JDBC DRIVERS

Client-side adaptors are called JDBC drivers that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

JDBC-ODBC bridge driver

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver makes the ODBC function calls from JDBC method calls. It is also called a Universal driver because it can be used to connect to any of the databases.

- As a common driver is used to interact with different databases, the data transferred through this driver is not so secure.
- The ODBC bridge driver is needed to be installed in individual client machines.
- The type-1 driver isn't portable.
- This driver software is built-in with JDK so no need to install it separately.
- It is a database-independent driver.

Native -API driver

The Native API driver uses the client-side libraries of the database. This driver form native calls of the database API from JDBC method calls. With the help of local API, this driver interacts with different databases, so

data transfer is more secure than the type-1 driver.

- The driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on the client machine.
- The type-2 driver isn't written in java, that's why it isn't a portable driver
- It is a database-dependent driver.

Network Protocol driver

The Network Protocol driver uses middleware (application server). And this middleware converts JDBC calls directly or indirectly into the vendor-specific database protocol. All the database connectivity drivers are present in a single server so no need for individual client-side installation.

- Type-3 drivers are fully written in Java, hence they are portable drivers.
- No client-side library is required because the application server can perform many tasks like auditing, load balancing, logging, etc.
- Network support is required on the client machine.
- Maintenance of the Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
- Switch facility to switch over from one database to another database.

Thin driver

The type-4 driver is also called the native protocol driver. This driver interacts directly with the database. It does not depend on any native database library, so it is also known as Thin Driver.

- Does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language, hence they are portable drivers.

Which Driver to use when?

- If you are accessing one type of database, then preferred 4 type driver.
- If you are accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 3 or type 4 driver is not available for your database, then type 2 drivers are useful.
- The type 1 driver is used for development and testing purposes only.

14.6 CONNECTION MANAGEMENT

Java Database Connectivity is done with the following five steps:

- Register for the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

1. Register for the driver class

The forName() method of Class is used to register the driver class. And it's used to dynamically load the driver class.

Syntax of forName() method:

```
public static void forName(String className)throws
ClassNotFoundException
```

2. Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method:

```
1) public static Connection getConnection(String
url)throws SQLException
2) public static Connection getConnection(String
url,String name,String password) throws SQLException
```

3. Create the Statement object

The createStatement() method of the Connection interface is used to create a statement. This statement is responsible to execute queries with the database.

Syntax of createStatement() method:

```
public Statement createStatement()throws SQLException
```

4. Execute the query

The executeQuery() method of the Statement interface is used to execute queries to the database. This method returns the object of ResultSet which consists of all records of a table.

Syntax of executeQuery() method:

```
public ResultSet executeQuery(String sql)throws
SQLException
```

5. **Close the connection object**

Closing connection object statement automatically closed the ResultSet. The close() method of Connection interface is used to close the connection.

Syntax of close() method:

```
public void close()throws SQLException
```



CHECK YOUR PROGRESS

Q 1. What are the packages related to JDBC classes?

Q 2. What are the steps related to JDBC process?

Q 3. How many drivers are available in JDBC?

Q 4. What is the use of JDBC API?

Q 5. How can you manage JDBC connection?

14.7 JDBC DESIGN CONSIDERATIONS

JDBC and database drivers work to access spreadsheets and databases. The design of JDBC defines the components of JDBC, which are used for connecting to the database.

Components of JDBC

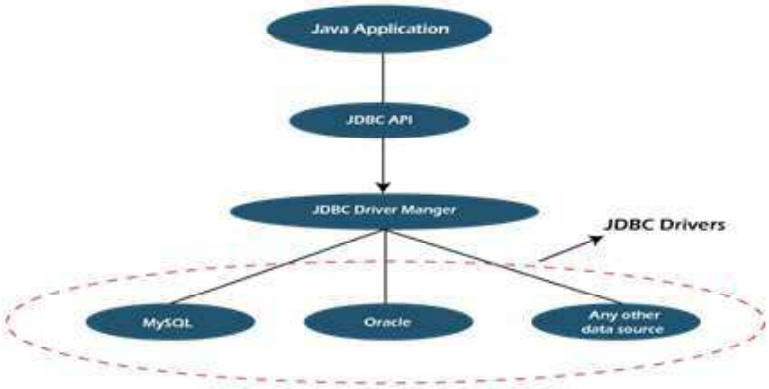


Figure 14.1: Component of JDBC

JDBC has four major components that are used for the interaction with the database.

1. JDBC API
2. JDBC Test Suite
3. JDBC Driver Manager
4. JDBC ODBC Bridge Driver

1) JDBC API: JDBC API provides various interfaces and methods to establish easy connections with different databases.

- javax.sql.*;
- java.sql.*;

2) JDBC Test suite: JDBC Test suite facilitates the programmer to check the various operations such as deletion, updating, and insertion which are executed by the JDBC Drivers.

3) JDBC Driver manager: JDBC Driver manager loads the database-specific driver into an application to establish the connection with the database. The JDBC Driver manager processes the request by converting the database-specific call to the database.

4) JDBC-ODBC Bridge Drivers: JDBC-ODBC Bridge Drivers establish the connection between the database drivers and the database. The bridge translates the JDBC method calls into the ODBC method call. It makes the usage of the native library to access the ODBC (Open Database Connectivity) characteristics.

14.8 TWO-TIER AND THREE-TIER CLIENT-SERVER MODEL

The JDBC architecture consists of two-tier and three-tier processing modes to access a database. They are as follows:

1. **Two-tier model:** In this architecture, a java application communicates directly with the data source. The JDBC driver establishes the communication between the application and the data source. The driver sent the answers for user queries back the result to the user after getting a query from the user to the data source.

The user can connect data sources that can be located on a different machine on a network. This is known as a **client/server configuration,** where the user's machine treat as a client, and the machine has the data source acting as the server.

2. **Three-tier model:** In this, the middleware services accept the user's queries from which the commands are again sent to the data source. The results are sent back to the middle tier, and the middle tier sends them to the user.

This type of model is found very useful by management information system directors.

14.9 RESULTSET

A ResultSet is the return data of the SQL statements that read data from a database query. The SELECT statement is the standard approach to select rows from a database and view them in a result set. The java.sql.ResultSet interface represents the result set of a database query.

A ResultSet object maintains a cursor to point to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface consist of three categories?

- Navigational methods ? Used to move the cursor around.
- Get methods ? Used to view the data in the columns of the current row being pointed by the cursor.
- Update methods ? Used to update the data in the columns of the current row from the underlying database.

To create statements with desired ResultSet, JDBC provides the following connection methods ?

- createStatement(int RSType, int RSConcurrency);
- prepareStatement(String SQL, int RSType, int RSConcurrency);
- prepare all(String sql, int RSType, int RSConcurrency); where the first argument shows the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is

read-only or updatable.

Type of ResultSet:

The possible ResultSet Types are as given below:

Table 14.1: Types of ResultSet

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

Important point: If you do not specify any ResultSet type, you will automatically get one that is TYPE_FORWARD_ONLY.

Viewing a Result Set:

The ResultSet interface contains methods for getting the data of the current row.

There is a get method for each of the possible data types with two versions:

- One that takes in a column name.
- One that takes in a column index.

For example, to view the column content which contains an int, need to use one of the getInt() methods of ResultSet -

```
public int getInt(String columnName) throws SQLException
returns the int in the current row in the column named columnName
```

Or

```
public int getInt(int columnIndex) throws SQLException
```

which returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

Updating a Result Set:

For updating the data of a result set, the ResultSet interface contains a collection of update methods. There are two update methods for each data type ?

- One that takes in a column name.
- One that takes in a column index.

For example, to update a String column of the current row of a result set, you can use one of the following updateString() methods -

```
public void updateString(int columnIndex, String s) throws SQLException
```

which changes the String in the specified column to the value of s.

or

```
public void updateString(String columnName, String s) throws SQLException
```

Similar to the previous method, except that the column is specified by its name instead of its index.

Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods.

Table 14.2: Different options for updating the row of ResultSet

S.N.	Methods & Description
1.**public void updateRow()**	Updates the current row by updating the corresponding row in the database.
2.**public void deleteRow()**	Deletes the current row from the database

3.**public void refreshRow()**	Refreshes the data in the result set to reflect any recent changes in the database.
4.**public void cancelRowUpdates()**	Cancels any updates made on the current row.
5.**public void insertRow()**	Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

Concurrency of ResultSet:

The possible ResultSetConcurrency are given below.

Table 14.3: Different options for Concurrency of ResultSet

Concurrency	**Description**
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default. It is the default setting for concurrency
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

Navigating a Result Set:

There are some methods to moving the cursor in the ResultSet are given below:

Table 14.4: Different options for Navigating a ResultSet

S.N.	**Methods & Description**
1	**public void beforeFirst() throws SQLException** Moves the cursor just before the first row.
2	**public void afterLast() throws SQLException** Moves the cursor just after the last row.
3	**public boolean first() throws SQLException** Moves the cursor to the first row.
4	**public void last() throws SQLException** Moves the cursor to the last row.
5	**public boolean absolute(int row) throws SQLException** Moves the cursor to the specified row.

6	**public boolean relative(int row) throws SQLException** Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
7	**public boolean previous() throws SQLException** Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
8	**public boolean next() throws SQLException** Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
9	**public int getRow() throws SQLException** Returns the row number that the cursor is pointing to.
10	**public void moveToInsertRow() throws SQLException** Moves the cursor to a special row in the result set that can be used to insert a new row into the database.
11	**public void moveToCurrentRow() throws SQLException** Moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing

14.10 PREPARED STATEMENT AND CALLABLE STATEMENT

14.10.1 Prepared Statement

To execute the precompiled SQL statement for more than one time, it is used statements extending from the PreparedStatement interface. This accepts parameterized SQL quires but user can pass 0 or more parameters to this query or we can pass '?' in initial stage. String sql="insert into emp values(?,?,?)"; in this example the argument '?' set by calling setter method.

Creating a PreparedStatement:

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Syntax for the PrepareStatement():

```
public PreparedStatement prepareStatement(String query)throws SQLException{}
```

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Table 14.5: Different methods from PreparedStatement Interface

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Executing the Prepared Statement:

After execution of prepareStatement() you can execute it using one of the execute() methods of the PreparedStatement interface

- **execute():** This method executes normal static SQL statements in the current prepared statement object and returns a boolean value.
- **executeQuery():** This method executes the current prepared statement and returns a ResultSet object.
- **executeUpdate():** This method executes SQL DML statements such as insert update or delete in the current Prepared statement. It returns

an integer value representing the number of rows affected.

Why use PreparedStatement?

The performance of the application will be improve after using PreparedStatement interface because query is compiled only once.

14.10.2 Callable Statement

The CallableStatement interface provides methods to execute the stored procedures. you can call stored procedures of all RDBMS in single standard way as the JDBC API provides a stored procedure SQL escape syntax. CallableStatement interface is used to call the stored procedures and functions. It gives better performance because these are precompiled.

Creating a CallableStatement:

To create an object of the CallableStatement (interface) we can use the prepareCall() method of the Connection interface. This method accepts a string variable representing a query to call the stored procedure and returns a CallableStatement object.

```
CallableStatement                    cstmt=
con.prepareCall("{callmyProcedure(?, ?, ?)}");
```

Execution of Callable Statement

The CallableStatement object you can execute it using following execute() method.

```
cstmt. execute();
```

28llo Java Program for Beginners

14.11 RESULTSET META DATA OBJECT

The metadata means data about data If you have to get metadata of a table like total number of column, column name, column type etc., ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Commonly used methods of ResultSetMetaData interface

Table 14.6: Different methods from ResultSetMetaData Interface

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.



CHECK YOUR PROGRESS

Q 6. What is the function of JDBC Driver Manager?

Q 7. What is the specialty of Three-tier architecture?

Q 8. What is ResultSet?

Q9. Write two differences between Statement and PreparedStatement interface?



14.12 LET US SUM UP

- ODBC abstraction is defined as C native functions. Therefore ODBC calls are C function calls.
- JDBC generally referred as Java Database Connectivity is one of the service API of Java platform.
- JDBC defines a standard protocol for Java Application to access tabular data stores.
- JDBC API needs to be supported by JDBC Driver to communicate wih the respective database.
- JDBC driver is a set of classes that implement the JDBC specification.
- JDBC drivers are responsible to drive the queries from java application to DBMS.

- There are 4 different drivers are related to JDBC. All have different responsibility.
- JDBC API is mainly divided into two packages jav.sql and javax.sql
- The statement object is used to execute the normal SQL statement without IN and OUT parameters.
- The resultset object is used to represent data the DB Cursor to java application, using JDBC to read the database.
- The PreparedStatement object is used to execute SQL statement with IN parameter. This supports the positional parameter, represent using '?'
- The callable statement object is used to execute Db stored procedure that supports IN and OUT parameters.
- The DatabaseMetaData object is used to get the database and JDBC driver details.



14.13 FURTHER READING

1) Seth, A. & Juneja, B. L. (2017), Java One Step Ahead(1st), Oxford University Press
2) Kumar, S. K.et al (2014) JDBC, Servlet, and JSP includes JSF and Design Patterns Black Book, DreamTech Press

14.14 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. Java.sql and javax.sql.

Ans to Q No 2. Import JDBC packages, Load and register the JDBC driver, Open a connection to the database, Create a statement object to perform a query, Execute the statement object and return a query resultset, Process the resultset, Close the resultset and statement objects, Close the connection.

Ans to Q No 3. There are four different drive are related to JDBC.

Ans to Q No 4. The JDBC API can virtually access any data source, from relational databases to flat file spreadsheets.

Ans to Q No 5. Register the Driver class, Create connection, Create statement, Execute queries, Close connection.

Ans to Q No 6. JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database. It process the user request.

Ans to Q No 7. In three-tier architecture, middleware services accept the user's queries from which the commands are again sent to the data source.

Ans to Q No 8. A ResultSet is the return data of the SQL statements that read data from a database query. The SELECT statement is the standard approach to select rows from a database and view them in a result set.

Ans to Q No 9. Statement used when SQL query is execute only once but Prepared statement executed when SQL query execute for multiple time.

In statement cannot parameter at runtime but it is possible in case of Prepared statement.



14.15 MODEL QUESTIONS

Q 1. What is JDBC Driver?

Q 2. What are the steps to connect to the database in java?

Q 3. What are the JDBC API components?

Q 4. What are the JDBC statements?

Q 5. What are the difference between PreparedStatement and CallableStatement?

Q 6. What is ResultSet?

Q 7. Describe is design structure of JDBC.

Q 8. What are the differences between execute, executeQuery, and executeUpdate?

Q 9. What is the difference between stored procedures and functions?

Q 10. Describe the design components of JDBC.

Q 11. Describe the two-tier and three-tier architecture of JDBC.

*** ***** ***

UNIT 15: DATABASE PROGRAMMING USING JAVA

UNIT STRUCTURE

15.1 Learning Objectives

15.2 Introduction

15.3 MySQL as Backend and Java As Frontend

- 15.3.1 JDBC Steps to Connect MySQLDatabase
- 15.3.2 Extract Data from MySQL Database
- 15.3.3 Create MySQLTable Using JDBC
- 15.3.4 Insert Data into Table
- 15.3.5 Update Data into Table
- 15.3.6 Delete Data from Table
- 15.3.7 Application of Java Swing and MySQL

15.4 Let Us Sum Up

15.5 Further Reading

15.6 Answer to Check Your Progress

15.7 Model Questions

15.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- connect JDBC with MySQL
- explain how to create database in MySQL
- explain how to extract data from MySQL table
- describe how to create table in MySQL through java programming
- describe how to insert and update data to MySQL table
- explain how to delete data from MySQL table
- describe application of user input though swing component and store into MySQL table.

15.2 INTRODUCTION

In the earlier unit, you have been introduced to JDBC. Topics like JDBC API and JDBC drivers along with the design and structure of JDBC have been described in the previous unit. In addition to these, concepts related to two-tier and three-tier architecture and ResultSet were also described in the previous unit.

This is the last unit of the course and in this unit, you will be introduced to database programming using JAVA. Topics like connecting to MySQL database and extracting data from MySQL databases are described in this unit. In addition to this, concepts related to creating a table, inserting, deleting and updating data from a table are also described in this unit.

15.3 MYSQL AS BACKEND AND JAVA AS FRONTEND

Practically MySQL is one of the RDBMS.DriverManager have JDBC driver which try to make Connections with MySQL database. The easiest way to connect is to use Class.forName() on the class that implements the java.sql.Driver interface. With MySQL Connector, the name of this class is com.mysql.jdbc.Driver. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

15.3.1 JDBC Steps to Connect MySQL Database

To connect Java application with the MySQL database, we need to go through the following steps.

- **Driver class:** The driver class com.mysql.jdbc.Driver use for the mysql database.
- **Connection URL:** The connection URL for the mysql database is jdbc:mysql://localhost:3306/DB where jdbc is the API, mysql is the database, localhost is the server name where mysql is running, 3306 is the port number and DB is the database name.

- **Username:** The default username for the mysql database is root.
- **Password:** The password choosen by the user at the time of installing the mysql database.

In MySQL, first create the database before creating table, to create database and table steps are as follows:

How to create database and table in MySQL

```
1. create database DB;
2. use DB;
3. create table emp(id int(10),name varchar(40),age
int(3));
```

Example 15.1: Example to open and close connected database

```
import java.sql.Connection;
import java.sql.DriverManager;
public class Connectiondata{
public static void main(String[] argv) throws
Exception{
String driver = "com.mysql.jdbc.Driver";
String connection = "jdbc:mysql://localhost:3306/
DBName";
String user = "root";
String password = "root";
Class.forName(driver);
Connection con = DriverManager.getConnection
(connection, user, password);
if(!con.isClosed()) {
con.close();
}
}
}
```

To connect java application with the mysql database, mysqlconnector.jar file is required to be loaded.

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1. Paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector .jar file and paste the jar file in jre/ lib/ext folder

2. Set classpath:

Set the classpath in a temporary or permanent way.

15.3.2 Extract Data FromMySQL Database

Example 15.2: Create a Connection of Java Application to MySQL database and read data from MySQL table data and display on the output screen.

```
import java.sql.*;
class MysqlConnect{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/DB","root","root");
//here root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"
"+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e)
{ System.out.println(e);}
}
}
```

15.3.3 Create MySQL Table Using JDBC

Example 15.3: Example to create MySQL table with java application package Edu;

```
import java.sql.*;
import java.sql.DriverManager;
public class Example {
static final String JDBC_DRIVER =
"com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://
localhost/emp";
static final String USER = "root";
static final String PASS = "";
public static void main(String[] args) {
Connection conn = null;
Statement stmt = null;
try{
//STEP 2: Register JDBC driver
Class.forName("com.mysql.cj.jdbc.Driver");
System.out.println("Connecting to database...");
conn =
DriverManager.getConnection(DB_URL,"root","");
System.out.println("Creating table in given
database...");
stmt = conn.createStatement();
String sql = "CREATE TABLE EMPLOYEES " +
"(id INTEGER not NULL, " +
" first VARCHAR(255), " +
" last VARCHAR(255), " +
" age INTEGER, " +
" PRIMARY KEY ( id ))";
stmt.executeUpdate(sql);
System.out.println("Created table in given
```

```
database...");
System.out.println("Inserting records into the
table...");
stmt =conn.createStatement();
String sql ="INSERT INTO Employees VALUES (100,
'Kriss', 'Kurian', 18)";
stmt.executeUpdate(sql);
sql = "INSERT INTO Employees VALUES (101, 'Enrique',
'John', 25)";
stmt.executeUpdate(sql);
sql= "INSERT INTO Employees (102, 'Taylor', 'Swift',
30)";
stmt.executeUpdate(sql);
sql= "INSERT INTO Employees VALUES(103, 'Linkin',
'Park', 28)";
stmt.executeUpdate(sql);
System.out.println("Inserted records into the
table...");
rs.close();
stmt.close();
conn.close();
}catch(SQLException se){
se.printStackTrace();
}catch(Exception e){
e.printStackTrace();
}finally{
try{
if(stmt!=null)
stmt.close()
}catch(SQLException se2){
}               // nothing can be done
try{
if(conn!=null)
```

```
conn.close();
}catch(SQLException se){
se.printStackTrace();
}
}
System.out.println("Goodbye!");
}
}
}
```



CHECK YOUR PROGRESS

Q 1. Describe the steps to establish JDBC connection.

Q 2. How can you create a table in MYSQL?

Q 3. How can you create an object for connection?

Q 4. Write SQL statement for insert data into table.

15.3.4 Insert Data Into Table

Example 15.4: Example of inserting Data into a MySQL table using PreparedStatement

```
import java.sql.*;
import java.util.Calender;
public class JavaMysqlPrStatementInsert{
public static void main(String[] args)
{ try{
String myDriver = "org.gjt.mm.mysql.Driver";
String myUrl = "jdbc:mysql://localhost/test";
Class.forName(myDriver);
Connection conn =
DriverManager.getConnection(myUrl, "root", "");
Calendar calendar = Calendar.getInstance();
```

```
java.sql.DatestartDate = new
java.sql.Date(calendar.getTime().getTime());
String query = " insert into users (first_name,
last_name, date_created, is_admin, num_points)"+"
values(?, ?, ?, ?, ?)";
PreparedStatementpreparedStmt =
conn.prepareStatement(query);
preparedStmt.setString (1, "Barney");
preparedStmt.setString (2, "Rubble");
preparedStmt.setDate (3, startdate);
preparedStmt.setBoolean(4, false);
preparedStmt.setInt (5, 5000);
preparedStmt.execute();
conn.close();
}
Catch( Exception e)
{
System.err.println("Got an exception!");
System.err.println(e.getMessage());
}
}
}
```

15.3.5 Update Data Into Table

Example 15.5: Example of Update Data into a MySQL table using PreparedStatement

```
Import java.sql.*;
Import java.util.Calender;
Public class JavaMysqlPrStatementUpdate{
public static void main(String[] args)
{ try {
String myDriver = "org.gjt.mm.mysql.Driver";
```

```
String myUrl = "jdbc:mysql://localhost/test";
Class.forName(myDriver);
Connection conn =
DriverManager.getConnection(myUrl, "root", "");
String query = "update users set num_points= ?
where first_name = ?";
PreparedStatementpreparedStmt =
conn.prepareStatement(query);
preparedStmt.setInt (1, 6000);
preparedStmt.setString (2, "XYZ");
preparedStmt.executeUpdate();
conn.close();
}
Catch( Exception e)
{
System.err.println("Got an exception!");
System.err.println(e.getMessage());
}
}
}
```

15.3.6 Delete Data From Table

Example 15.6: Example to delete record from table in MySQL Database using PreparedStatement

```
package com.boraji.tutorial.jdbc;
importjava.sql.Connection;
importjava.sql.DriverManager;
importjava.sql.PreparedStatement;
importjava.sql.SQLException;
publicclassDeleteRecordExample{
publicstaticvoidmain(String[] args) {
String jdbcUrl = "jdbc:mysql://localhost:3306/DB";
```

```
String username = "root";
String password = "root";
String sql = "delete from employee where emp_id=?";
try
(Connection conn =
DriverManager.getConnection(jdbcUrl,username,
password);
PreparedStatementstmt = conn.prepareStatement(sql))
{ stmt.setInt(1, 2);
stmt.executeUpdate();
System.out.println("Record deleted successfully");
}
catch (SQLException e)
{ e.printStackTrace();
 }
}
}
```

15.3.7 Application Of Java Swing And MySQL

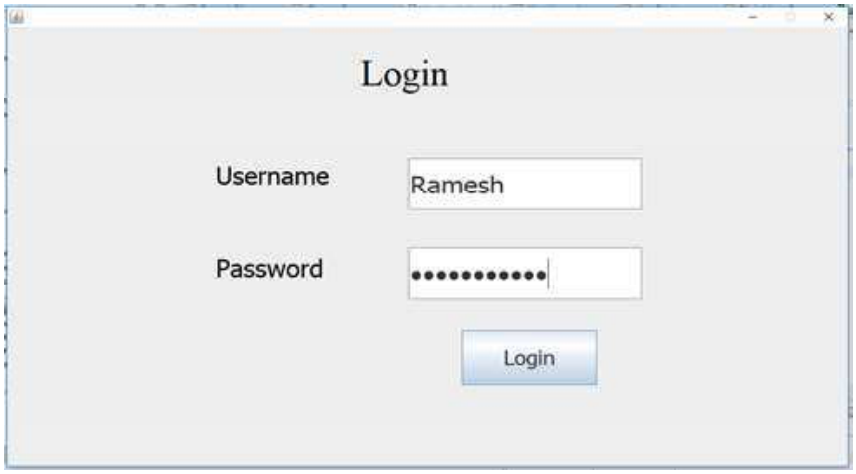


Figure 15.1: Example of user login page

Program 15.1: Create a simple java program to user login form using Swing APIs.

```
packagecom.javaguides.javaswing.login;
importjava.awt.Color;
importjava.awt.EventQueue;
importjava.awt.Font;
importjava.awt.event.ActionEvent;
importjava.awt.event.ActionListener;
importjava.sql.Connection;
importjava.sql.DriverManager;
importjava.sql.PreparedStatement;
importjava.sql.ResultSet;
importjava.sql.SQLException;
importjavax.swing.JButton;
importjavax.swing.JFrame;
importjavax.swing.JLabel;
importjavax.swing.JOptionPane;
importjavax.swing.JPanel;
importjavax.swing.JPasswordField;
importjavax.swing.JTextField;
importjavax.swing.border.EmptyBorder;
publicclassUserLoginextendsJFrame {
privatestaticfinallongserialVersionUID=1L;
privateJTextFieldtextField;
privateJPasswordFieldpasswordField;
privateJButtonbtnNewButton;
privateJLabel label;
privateJPanelcontentPane;
publicstaticvoidmain(String[] args) {
EventQueue.invokeLater(newRunnable()  {
publicvoidrun() {
try  {
```

```
        UserLogin frame =newUserLogin();
        frame.setVisible(true);
        } catch (Exception e) {
        e.printStackTrace();
        }
        }
        });
        }
        publicUserLogin()     {

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setBounds(450, 190, 1014, 597);
        setResizable(false);
        contentPane=newJPanel();
        contentPane.setBorder(newEmptyBorder(5, 5, 5,
5));
        setContentPane(contentPane);
        contentPane.setLayout(null);
        JLabellblNewLabel=newJLabel("Login");
        lblNewLabel.setForeground(Color.BLACK);
        lblNewLabel.setFont(newFont("Times  New
Roman", Font.PLAIN, 46));
        lblNewLabel.setBounds(423, 13, 273, 93);
        contentPane.add(lblNewLabel);
        textField=newJTextField();
        textField.setFont(newFont("Tahoma",
Font.PLAIN, 32));
        textField.setBounds(481, 170, 281, 68);
        contentPane.add(textField);
        textField.setColumns(10);
        passwordField=newJPasswordField();
```

```
        textField.setFont(newFont("Tahoma",
Font.PLAIN, 32));
        textField.setBounds(481, 170, 281, 68);
        contentPane.add(passwordField);
        JLabellblUsername=newJLabel("Username");
        lblUsername.setBackground(Color.BLACK);
        lblUsername.setForeground(Color.BLACK);
        lblUsername.setFont(newFont("Tahoma",
Font.PLAIN, 31));
        lblUsername.setBounds(250, 166, 193, 52);
        contentPane.add(lblUsername);
        JLabellblUsername=newJLabel("Password");
        lblUsername.setBackground(Color.BLACK);
        lblUsername.setForeground(Color.CYAN);
        lblUsername.setFont(newFont("Tahoma",
Font.PLAIN, 31));
        lblUsername.setBounds(250, 166, 193, 52);
        contentPane.add(lblPassword);
        btnNewButton=newJButton("Login");
        btnNewButton.setFont(newFont("Tahoma",
Font.PLAIN, 26));
        btnNewButton.setBounds(545, 392, 162, 73);
        btnNewButton.addActionListener(newActionListener()
{
        publicvoidactionPerformed(ActionEvente)
{
        StringuserName=textField.getText();
        String password =passwordField.getText();
        try  {
        Connectionconnection= (Connection)
        DriverManager.getConnection("jdbc:mysql://
localhost:3306/swing_demo" "root", "root");
```

```
      PreparedStatementst=  (PreparedStatement)
connection
      .prepareStatement("Select name, password from
student where name=? and password=?");
      st.setString(1, userName);
      st.setString(2, password);
      ResultSetrs=st.executeQuery();
      if (rs.next())   {
      dispose();
      UserHome ah =newUserHome(userName);
      ah.setTitle("Welcome");
      ah.setVisible(true);
      JOptionPane.showMessageDialog(btnNewButton,
"You have successfully logged in");
      }     else  {
      JOptionPane.showMessageDialog(btnNewButton,
"Wrong Username & Password");
      }
      }
      catch (SQLExceptionsqlException) {
      sqlException.printStackTrace();
      }
      }
      });
      contentPane.add(btnNewButton);
      label =newJLabel("");
      label.setBounds(0, 0, 1008, 562);
      contentPane.add(label);
      }
      }
```



CHECK YOUR PROGRESS

Q 5. Write SQL query toupdate data into MySQL table?

Q 6. Write SQL query to delete data from MySQL table?

Q 7. What is the use of the following code?

```
btnNewButton.setFont(newFont("Tahoma", Font.PLAIN,
26))
```



15.4 LET US SUM UP

- To connect Java application with the MySQL database, we need to go through the following parts: driver, connection URL, user and password.
- We can create table in MySQL using Java programming.
- We can insert, update data into MySQL table using java coding.
- Deletion of record is possible through JBDC connectivity.
- Different component like java swing also applicable to taking the user input and store data to database table.



15.5 FURTHER READING

1) Kumar, S. K.et al (2014), JDBC, Servlet, and JSP includes JSF and Design Patterns Black Book, DreamTech Press

2) Sharan, K.et al.(2022), Beginning Java 8 APIs, Extensions and Libraries, 1st ed., Apress Berkeley, CA



15.6 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. The steps to establish JDBC connection are:

- define driver
- define connection URL
- username
- password

Ans to Q No 2. The steps to create MySQL table are:

```
create database DB;
use DB;
create table emp(id int(10),name varchar(40),age int(3));
```

Ans to Q No 3. The query for creating an object for connection is:

```
String sql ="INSERT INTO Employees VALUES (100,
'Kriss', 'Kurian', 18)";
```

Ans to Q No 4. The SQL statements for inserting data into table are:

```
String myDriver = "org.gjt.mm.mysql.Driver";
String myUrl = "jdbc:mysql://localhost/test";
Class.forName(myDriver);
Connection conn = DriverManager.getConnection(myUrl,
"root", "");
```

Ans to Q No 5. The string query to update data into MySQL table is:

```
"update users set num_points= ? where first_name =
?";
```

Ans to Q No 6. The query to delete data from MySQL table is:

```
String sql = "delete from employee where emp_id=?";
```

Ans to Q No 7. The use of the code is to set font properties for a button.



15.7 MODEL QUESTIONS

Q 1. Write how to create JDBC connectivity.

Q 2. Write about the important points related to the JDBC connection.

Q 3. Write MySQL coding to create a table and display a record from the table.

Q 4. Write a Java program to create tables related to your student information at your university.

Q 5. Write a Java program to extract data from the student table.

Q 6. Write a Java program to enter new data into your student table using java swing.

Q 7. Write a Java program to enter updated data into your student table from user input.

Q 8. Write a Java program to create tables to store exam-related information for students.

Q 9. Write a Java program to enter delete records from your student after passing out of the particular student.

*** ***** ***