PGSc IT S4 01
Exam. Code: M4PIJ

Programming in Java

SEMESTER- IV

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY

BLOCK-1

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

Programme Coordinator : Dr. Ridip Dev Choudhury, KKHSOU

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

Editorial Team

Content & Language: Dr. Ridip Dev Choudhury, KKHSOU

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

September, 2022 **ISBN:**

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
Block-1
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**

COURSE INTRODUCTION

This is a course on Programming in Java. 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.

The course consists of two blocks.

Block 1 deals with the basic concepts of the Java programming language. The operators and assignments used in Java are discussed with examples. The concept of Object Oriented Programming is also explained in this block. The uses of arrays, strings and vectors in Java are explained. At the end, this block introduces the concepts of interfaces, packages and threads.

Block 2 discusses about the uses of Java.lang.Math class at the beginning. It also discusses the process of exception and file handling mechanism used in Java, how to use applets, AWT and swings. In the last part of this block, the concept of JDBC and database programming are explained.

BLOCK INTRODUCTION

This is the first block of the course ***Programming in Java.*** This block contains eight units.

Unit 1 describes the features of Java, Java environment and installation process of Java SDK.

Unit 2 discusses about the tokens, keywords, constants, data types, uses of variables, and various decision and loop statements used in Java.

Unit 3 discusses the role of system analyst in the design and development process of a system.

Unit 4 describes the uses of classes in Java.

Unit 5 describes various types of inheritance, super class and static members used in Java.

Unit 6 discusses the uses of arrays, strings and vectors.

Unit 7 discusses the concept and uses of interfaces and packages.

Unit 8 deals with the various topics related to threads.

Each unit of these blocks includes "CHECK YOUR PROGRESS" questions. These have been designed 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

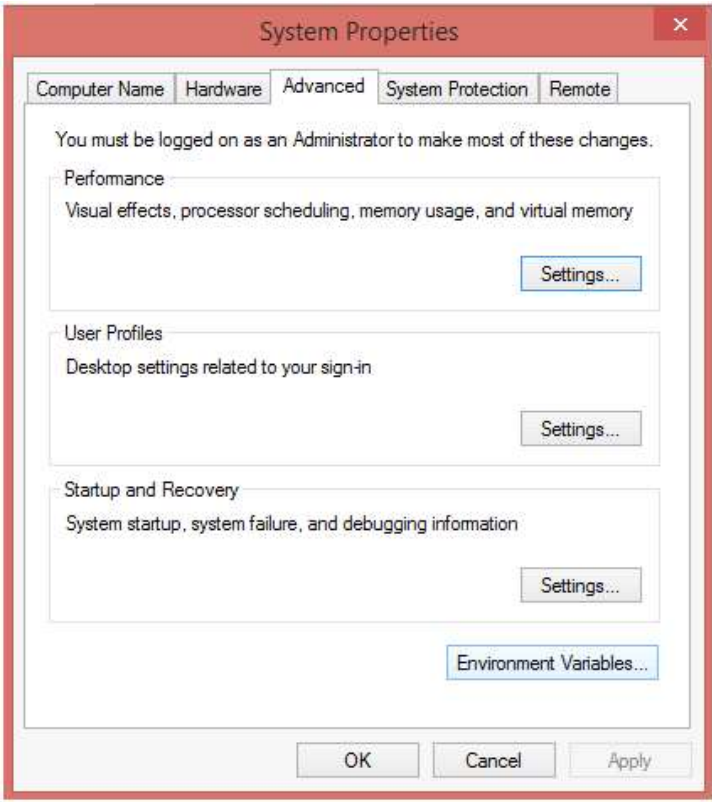
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

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

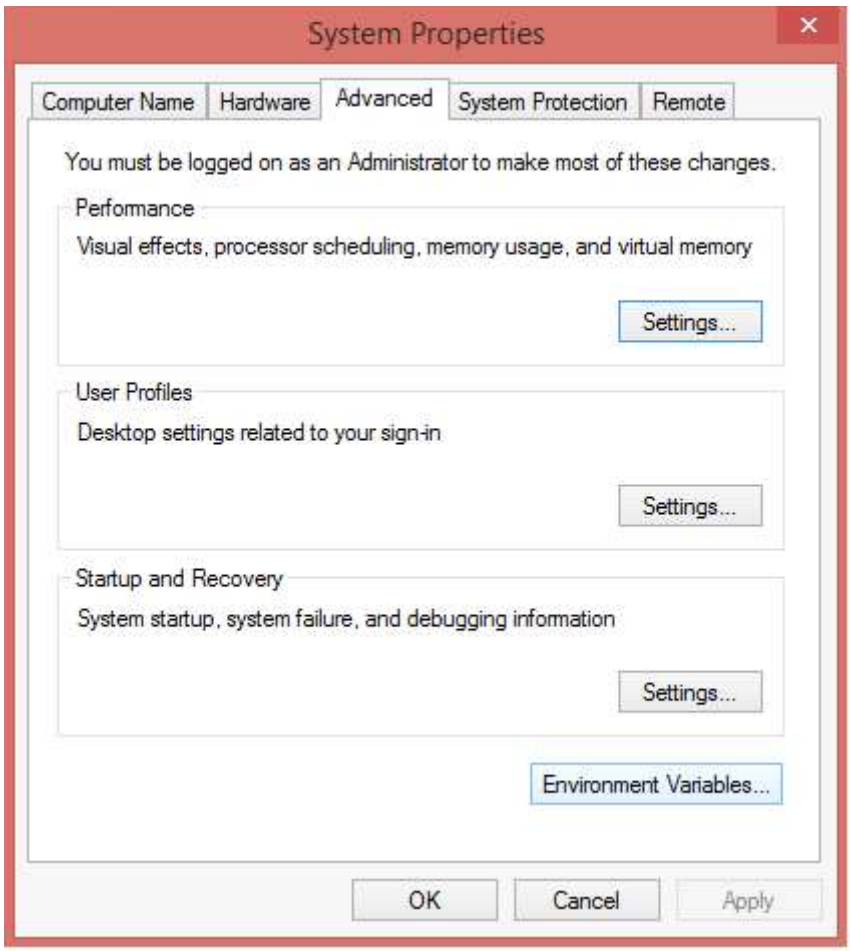
Linux macOS Windows

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");
  }
}
```

2.6 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?

.....

2.7 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);
      }
}
```

2.8 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)

2.9 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]

2.10 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

2.11 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);



2.12 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.



2.13 FURTHER READINGS

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



2.14 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



2.14 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

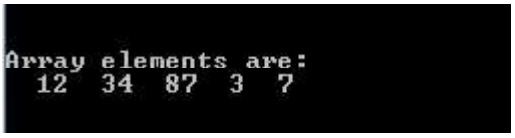
Data_Type Array_Name [ ]={ Data1, Data2, Data3,Data4,……..};

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:

```
Enter number of rows:
2
Enter number of columns:
3


Enter array elements:
Enter (0,0)th element:
12
Enter (0,1)th element:
6
Enter (0,2)th element:
8
Enter (1,0)th element:
43
Enter (1,1)th element:
22
Enter (1,2)th element:
78

Array elements are:
  12  6  8

  43  22  78
```

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.

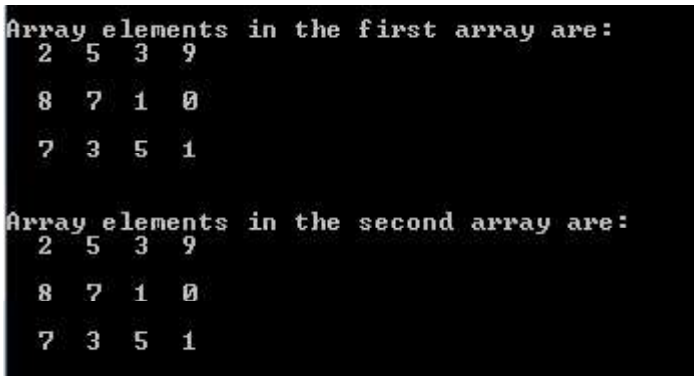
Example: Java program to initialize two-dimensional array

```
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];
```

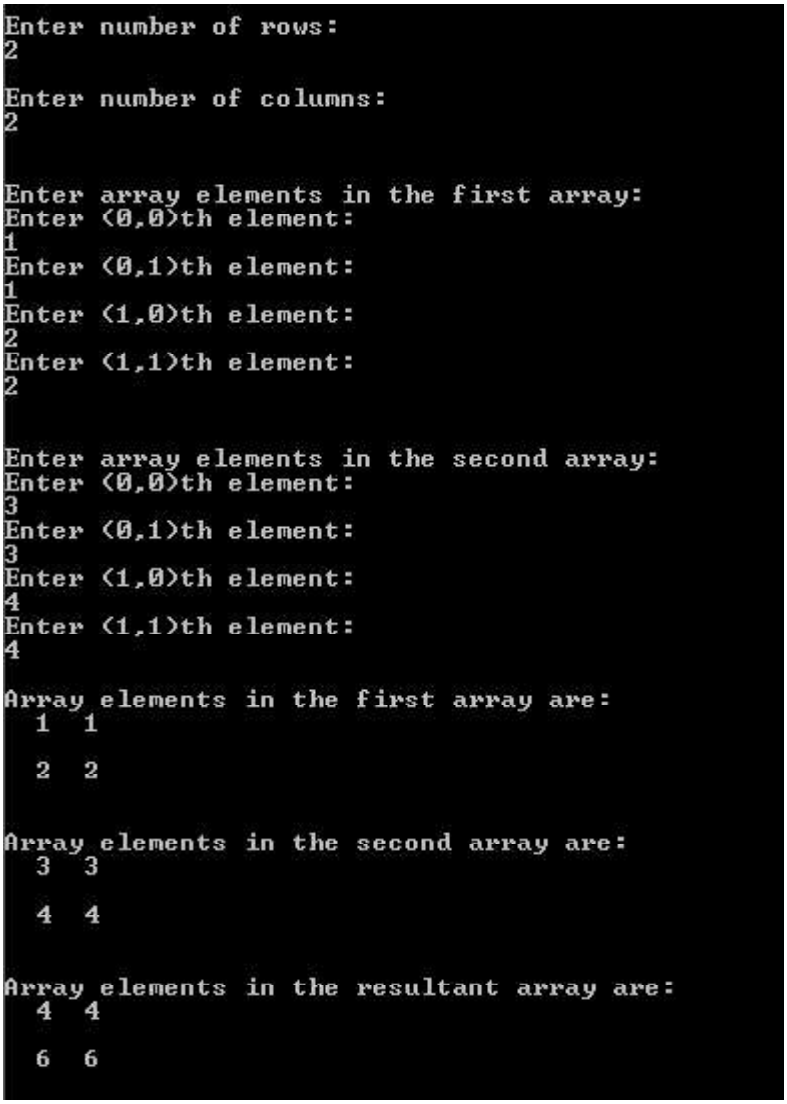
```
//Resultant two-dimensional array creation in memory
 arrResult = new int[rowNo][colNo];
System.out.println("\n\nEnter array elements in the
first array:");
// Insertion of integer values into the first two-
dimensional array
for(int i=0;i<rowNo;i++)
{
     for(int j=0;j<colNo;j++)
     {
        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

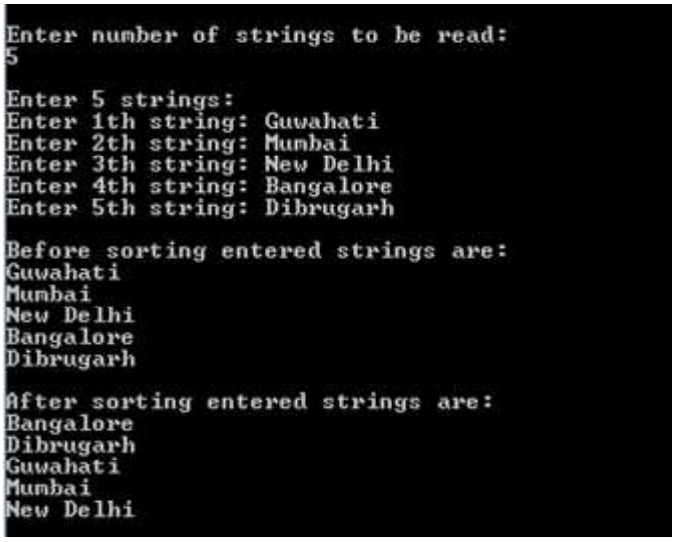
The following program demonstrates the use of some String class methods.

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.

*** ***** ***