UNIT 1

Object Oriented programming and Java

Structure

1.0 Introduction to Java
1.1 Introduction to byte code
1.2 JVM, JRE and JIT Compiler
1.3 Java and Internet
1.4 Platform independence of Java
1.5 Introduction to Applets
1.6 Main features of java
1.7 Introduction to OOPS
1.8 Main features of OOPS
1.9 Classes and Objects
1.10 public, protected and private
1.11 Simple Examples of objects and classes
1.12 Installing and configuring Java

Learning Objectives

After studying this unit, the student will be able to
• What is a java , Java program , compilation of java program, compilers
• Java linked with internet, platform independent, main features of java
• OOPS, classes and objects, simple example of classes

1.0 Introduction to Java

Java has evolved to be the most predominant and popular general purpose programming language of the current age. Java is a simple, portable, distributed, robust, secure, dynamic, architecture neutral, object oriented programming language. Java was originally developed by James Gosling at Sun Microsystems in 1995. This technology allows the software designed and developed once for an idealized ‘virtual machine’ and run on various computing platforms.

Java plays a significant role in the corporate world. Companies of all sizes are using Java as the main programming language to develop various applications/projects worldwide. It has found its use in various sectors including banking, insurance, retail, media, education, manufacturing and so on. E-commerce, Gaming, Mobile, Embedded, Media and many more types of applications are being developed using Java. Organizations are developing mission critical applications using Java technologies. This necessitates the corporations to hire highly skilled java developers. On the other hand it opens the doors for many opportunities for java developers. There is significant demand for java developers with sound technical skills. Now Colleges and Universities all over the world are using it in their introduction courses as well as their junior and senior software engineering courses.

First program

We will learn how to write a simple java program. Creating hello java example is too easy. Here we have created a class named simple that contains only main method and prints a message hello java. It is a simple java program. It is very important to note that the java programs are case sensitive.

For execution of java program

• Install the JDK
• Set path of the bin directory under JDK
• Create and edit the java program by using any text editor such as notepad, msword,etc
• Compile and execute the program
Creating hello java Example

class Simple{
    public static void main(String args[]){
        System.out.println(“Hello Java”)
    }
}

Save the file as simple.java

To compile: Javac simple.java

To execute: Java simple

Output: Hello Java

Understanding the first java program

Let us see what are the meanings of class, public, static, void, main, String[], System.out.println().

• Class is used to declare a class in java

• Public is an access modifier which represents visibility, it means it is visible to all.

• Static is a keyword, if we declare any method as a static, it is known as static method.

    The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it does not require to create an object to invoke the main method. So it saves memory.

    • Void: Is the return type of the method, it means it doesn’t return any value.

    • Main: Represents the startup of a program

    • String[] args: Is used for command line argument.

    • System.out.println(): Is a print statement.
What happens when the program is compiled

At compile time, java file is compiled by java compiler, (it does not interact with OS) and converts the java code into bytecode. Here simple.java will be compiled as simple.class

### 1.1 Introduction to bytecode

Java bytecode is the form of instructions that the Java virtual machine executes. Each bytecode opcode is one byte in length, although some require parameters, resulting in some multi-byte instructions. Not all of the possible 256 opcodes are used. 51 are reserved for future use. Beyond that, Sun Microsystems, the original creators of the Java programming language, the Java virtual machine and other components of the Java Runtime Environment (JRE), have set aside three values to be permanently unimplemented.

A Java programmer does not need to be aware of or understand Java bytecode at all. However, as suggested in the IBM developerWorks journal, “Understanding bytecode and what bytecode is likely to be generated by a Java compiler helps the Java programmer in the same way that knowledge of assembly helps the C or C++ programmer.”

### 1.2 JVM, JRE and JIT Compiler

#### 1.2.1 Java virtual machine (JVM)

A Java virtual machine (JVM) is a virtual machine that can execute Java bytecode. It is the code execution component of the Java software platform.

#### 1.2.2 Java Runtime Environment (JRE)

The Java Runtime Environment (also called JRE) is a software framework developed by Oracle Corporation, which runs regardless of the computer architecture. It provides the Java virtual machine and a large library, which allows to run applications written in the Java programming language.
It is constituted by

- A Java virtual machine. The heart of the Java platform is the concept of a “virtual machine” that executes Java bytecode programs. This bytecode is the same no matter what hardware or operating system the program is running under.

- Its associated Java Class Library, which is the Standard library available to Java programs.

1.2.3 JIT compiler

In computing, just-in-time compilation (JIT), also known as dynamic translation, is a method to improve the runtime performance of computer programs based on byte code (virtual machine code). Since byte code is interpreted it executes more slowly than compiled machine code, unless it is actually compiled to machine code, which could be performed before the execution – making the program loading slow – or during the execution. In this latter case – which is the basis for JIT compilation – the program is stored in memory as byte code, but the code segment currently running is preparatively compiled to physical machine code in order to run faster.

1.3 Java and Internet

Java is strongly associated with the internet because of the fact that the first application program written in Java was HotJava, a webbrowser to run applets on internet. Internet users can use java to create applet programs and run them locally using a “Java enabled browser” such as HotJava. They can also use Java enabled browser to download an applet located on a computer anywhere in the internet and run it on his local computer.

1.4 Platform independence of java

Platform independent: Unlike many other programming languages including C and C++ when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run. Platform means an operating system such as windows, unix, xenix, etc.,

It promised Write Once, Run Anywhere (WORA), providing no-cost run-times on popular platforms.
1.5 Introduction to Applets

Applets are small Java programs that are primarily used in internet computing. They can be transported over the internet from one computer to another and run using AppletViewer or by any Webbrowser that supports Java. An applet, like any Application program, can do many things for us. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games etc.

1.6 Main Features of Java

Object Oriented: In Java everything is an Object. Java can be easily extended since it is based on the Object model.

Platform independent: Unlike many other programming languages including C and C++ when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.

Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master.

Secure: With Java’s secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architectural-neutral: Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence Java runtime system.

Portable: We may carry the java bytecode to any platform.

Robust: Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multi-threaded: With Java’s multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.

High Performance: With the use of Just-In-Time compilers Java enables high performance.
Distributed: Java is designed for the distributed environment of the internet.

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

1.7 Introduction to OOP

Object Oriented Programming or OOP is the technique to create programs based on the real world. Unlike procedural programming, here in the OOP programming model programs are organized around objects and data rather than actions and logic. Objects represent some concepts or things and like any other objects in the real world, objects in programming language have certain behavior, properties, type, and identity. In OOP based language the principal aim is to find out the objects to manipulate and their relation between each other. OOP offers greater flexibility and compatibility and is popular in developing larger application. Another important work in OOP is to classify objects into different types according to their properties and behavior. So OOP based software application development includes the analysis of the problem, preparing a solution, coding and finally its maintenance.

Java is an object oriented programming and to understand the functionality of OOP in Java, we first need to understand several fundamentals related to objects. These include class, method, inheritance, encapsulation, abstraction, polymorphism etc.

1.8 Main features of OOPS

There are four main features of OOPS are

1. Abstraction
2. Encapsulation
3. Polymorphism
4. Inheritance

1. Abstraction: The process of abstraction in Java is used to hide certain details and only show the essential features of the object. In other words, it deals with the outside view of an object (interface).

2. Encapsulation: The process of wrapping data into a single unit is called Encapsulation.
3. **Inheritance**: This is the process in which a property of a predefined class can be inherited in a new class making the object of that class in the making class.

4. **Polymorphism**: This is the process in which a program can have more than one function with the same name but different parameters.

The concept of abstraction and encapsulation can be well understood by considering the simple examples. When we drive a motorbike, to stop our bike, simply we press the brakes. When we press the brakes, then the bike will stop. But we don’t know the internal mechanism that how it is stopped when we press the brakes because we are **abstracted** from the internal mechanism which a bike is **encapsulated** with a systematic mechanism.

One more example that the mobile phone is **encapsulated** (packaging) with the electronic circuitry with many features such as messages, calls, games, tools, music, internet etc.,. But to make a phone call, simply we press green button and after our conversation simply we press red button to stop. We know only simple things to start and stop because we are **abstracted** from the internal circuitry of mobile. But there is a large internal process of sending and receiving information through different networks and different places with electronic circuitry.

### 1.9 Classes and Objects

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. Class is a logical entity and object is a physical entity. OOPS simplifies the software development and maintenance by providing some concepts.

- Class
- Object

#### 1.9.1 Class

A class is a group of objects that gave common property. It is a template or blueprint from which objects are created.

**A class in java can contain**

- Data member
- Method
- Constructor
- Block
syntax of a class

```java
class <class_name>{
    data member;
    method;
}
```

Example for a class, suppose there is a program test.java

```java
class test{
    public static void main(String args[]){
        System.out.println("Welcome to II CSE Students");
    }
}
```

The output of the program is

Welcome to II CSE Students

### 1.9.2 Object

A runtime entity that has state and behavior is known as an object. For example a pen, a car, a man etc are the objects. It can be tangible or intangible (physical or logical). An object has three characteristics.

- **State**: Represents the data of an object
- **Behavior**: Represents the behavior of an object
- **Identity**: Identity is typically implemented via a unique ID. The value of the ID is not visible to the external user, but is used internally by the JVM to identify each object uniquely.

For example Pen is an object, Its name is Parker, color is blue etc., known as its state. It is used to writing is its behavior. Object is an instance of a class. Class is a template or blue print from which objects are created. So object is the instance of a class.

### 1.10 Classes: public, protected and private

Almost every thing is a class in Java, except for primitives such as ints, chars, doubles, etc. and all classes (reference types according to the lang spec) are derived from Object.

The cornerstone of most Object Oriented programming languages are classes.
Java has four types of access levels and one unnamed default one

- **Public**: Accessible everywhere.
- **Private**: Accessible only within the class
- **Protected**: Accessible to those within the same file/package and/or derived classes.
- **Private protected**: Accessible to those within the class and derived classes.

The default access level is very similar to protected.

1.11 Simple example of objects and classes

In this example, we have created a student class that have two data members `empid` and `empname`. We are creating the object of the student class by new keyword and printing the objects value.

```java
class Student{
    int empid; // data member or instance variable
    String empname; // data member or instance variable
    public static void main(String Args[]){
        Student s1 = new Student(); // creating an object of student
        System.out.println(s1.empid +" " +s1.empname);
    }
}
```

**output**

0 null

**Instance variable**

A variable that is created inside the class is also known as instance variable. These variables are only declared and therefore no storage space has been created in the memory. Instance variables are also known as member variables.

**Method**

In java, a method is like a function i.e, used to expose behavior of an object.
Advantage of method

- Code reusability
- Code optimization

New keyword

The new keyword is used to allocate memory at runtime.

Example of object and class that maintains the records of students

In this example, we are creating the three objects of Student class and initializing the values to these objects by invoking the InsertRecord method on it. Here, we are displaying the state(data) of the objects by invoking the DisplayInformation method.

```java
class Student{
    int rollno;
    String name;

    void InsertRecord(int r, String n) { //method
        rollno = r;
        name = n;
    }

    void DisplayInformation(){
        System.out.println(rollno+ " "+name); }

    public static void main(String args[]){
        Student s1 = new Student();
        Student s2 = new Student();
        Student s3 = new Student();
        s1.InsertRecord(101,"Rama");
        s2.InsertRecord(102,"Surya");
        s3.InsertRecord(101,"Chandra");
    }
}
```
s1.DisplayInformation();  
s2.DisplayInformation();  
s3.DisplayInformation();  
}  
}

The output of the above program is

101  Rama
102  Surya
101  Chandra

**Another example of object and class**

class Rectangle{
    int length;
    int width;

    void insert(int i ,int w){
        length = i;
        width = w; }

    void CalculateArea(){
        System.out.println("Area = "+length*width);}
    public static void main(String args[]){
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();
        r1.insert(12,3);
        r2.insert(15,4);
        r1.CalculateArea();
        r2.CalculateArea();} }
The output of the above program is

Area = 36
Area = 60

The first step in the installation of Java is to download the Java Development Kit (JDK) from the java.sun.com website. It is free software to download. We must choose the right version of JDK to the configuration of your system (operating system, 32 or 64 bit etc.,) at the time of download. Once the setup file is downloaded, we may proceed to install Java.

### 1.12 Installing and configuring Java

To install Java, we need to perform the following steps

1. Double click the .exe file to initiate the installation procedure. The screen appears as shown in Fig 1.1.

![Fig 1.1](Image)

2. After then, the welcome screen appears as shown Fig 1.2

![Fig 1.2](Image)
3. In the welcome screen, if we click on Next, starts the installation of java. Once the installation is completed, the complete screen appears as shown in Fig 1.3

4. The screen describes the successful installation of Java on the computer system.

   Click the close button to end the installation process.

**Configuring Java**

Once the Java is installed, we need to configure it by adding the java path to the environment variable, PATH. This eliminates the restriction to use java only from its current working directory. To set the PATH variable to the java directory, we need to perform the following steps.

1. Right click on the “My Computer” or “Computer” icon as shown in Fig 1.4
And select the properties option from the drop down menu. The system properties dialogue box appears as shown in Fig 1.5

![Fig 1.5](image)

Select the “advanced system settings” option to display the advanced page as shown in Fig 1.6

![Fig 1.6](image)
Click the “Environment Variables” button to display the Environment variables dialogue box as shown in Fig 1.7

![Environment Variables Dialogue Box](image1.png)

**Fig 1.7**

The environment variable dialogue box is divided into two sessions

- User variables
- System variables

Under system variable section, select the path option below the variable column and click the Edit button. The Edit system variable dialogue box appears as shown in Fig 1.8

![Edit System Variable Dialogue Box](image2.png)

**Fig 1.8**
By default, the path variable is already set to multiple locations. To set the Java directory path to the Path variable, append the directory path in the variable value text box, separated by a semi-colon, as shown in Fig 1.9.

![Edit System Variable](image)

Click OK to close the Edit system Variable dialogue box.

Click OK to close Environment Variable dialogue box.

Click OK to close System Properties dialogue box and complete the process of configuring Java.

**Summary**

Java programs are case sensitive. Lowercase characters are not equivalent to uppercase characters and vice versa. Suppose a class name `CalculatingArea` is not equivalent to `calculatingarea`.

**Short Answer Type Questions**

1. Who developed Java language.
2. Write a simple Java program to display “hello world”.
3. What are the commands for compilation and execution of Java programs?
4. What is Java bytecode?
5. What is JVM?
6. What is JRE?
7. What is JIT compiler?
8. How Java is platform independent.
9. What are applets?
10. What is a class?
11. What is an object?
12. What is an instance variable?
13. What are advantages of a method?
14. What is the expansion of WORA?

Long Answer Type Questions

1. Explain briefly class, public, static, void, main, string[], and system.out.println().
2. Write about the main features of java.
3. Write about the main features of OOPS.
4. A Record contains employee number (empno), employee name (ename) and salary (sal). So, write a program to display the details of three employees.
Structure

2.0 Introduction
2.1 Tokens
2.2 Data Types
2.3 Operators in Java
2.4 Simple methods
2.5 Type conversion and casting
2.6 Constructors and destructors

Learning Objectives

After studying this unit, you will be able to understand

- Various data types
- Operators in java
- Creating methods and calling them
- Conversion of data types and casting
- Constructors and destructors
2.0 Introduction

Java Tokens: A Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements. Most statements contain expressions, which describe the actions carried out on data. Smallest individual units in a program are known as tokens.

2.1 Tokens

In simple terms, a Java program is a collection of

(a) Tokens,
(b) Comments and
(c) White spaces.

(a) Tokens

Java language includes five types of tokens.

1. Reserve Word or Keywords
2. Identifier
3. Literals
4. Operators
5. Separators

1. Reserved words (Keywords)

There are certain words with a specific meaning in Java which tell (help) the compiler what the program is supposed to do. These Keywords cannot be used as variable names, class names, or method names. Keywords in Java are case sensitive, all characters being lower case.

Keywords are reserved words that are predefined in the language; see the table below (Taken from Sun Java Site). All the keywords are in lowercase.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Keyword</th>
<th>Keyword</th>
<th>Keyword</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>Default</td>
<td>If</td>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>Do</td>
<td>Implements</td>
<td>Protected</td>
<td>Throw</td>
</tr>
<tr>
<td>Break</td>
<td>Double</td>
<td>Import</td>
<td>Public</td>
<td>Throws</td>
</tr>
<tr>
<td>Byte</td>
<td>Else</td>
<td>Instanceof</td>
<td>Return</td>
<td>Ransient</td>
</tr>
<tr>
<td>Case</td>
<td>Extends</td>
<td>Int</td>
<td>Short</td>
<td>Try</td>
</tr>
</tbody>
</table>
Identifiers are the names of variables, methods, classes, packages and interfaces. Unlike literals they are not the things themselves, just ways of referring to them. In the HelloWorld program, HelloWorld, String, args, main and println are identifiers.

Identifiers must be composed of letters, numbers, the underscore_ and the dollar sign $. Identifiers may only begin with a letter, the underscore or a dollar sign.

Each variable has a name by which it is identified in the program. It’s a good idea to give your variables mnemonic names that are closely related to the values they hold. Variable names can include any alphabetic character or digit and the underscore _. The main restriction on the names you can give your variables is that they cannot contain any white space. You cannot begin a variable name with a number. It is important to note that as in C but not as in Fortran or Basic, all variable names are case-sensitive. MyVariable is not the same as myVariable. There is no limit to the length of a Java variable name. The following are legal variable names.

- My Variable
- Myvariable
- My Variable
- X
- I
- Myvariable
- $myvariable
- 9pins
- Student
The following are not legal variable names

- My Variable // Contains a space
- 9pins // Begins with a digit
- a+c // The plus sign is not an alphanumeric character
- Testing1-2-3 // The hyphen is not an alphanumeric character
- O’Reilly // Apostrophe is not an alphanumeric character
- a&b // ampersand is not an alphanumeric character

3. Literals

Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables. Java language specifies five major types of literals. They are

- Integer literals
- Floating point literals
- Character literals
- String literals
- Boolean literals

Each of them has a type associated with it. The type describes how the value behave and how they are stored.

4. Operators

An operator is a symbol that takes one or more arguments and operates on them to produce a result.

5. Separators: Separators help define the structure of a program.

The table lists the six Java separators.

<table>
<thead>
<tr>
<th>Separators</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses ()</td>
<td>Encloses arguments in method definitions and calling; adjusts precedence in arithmetic expressions; surrounds cast types and delimits test expressions in flow control statements.</td>
</tr>
<tr>
<td>Braces {}</td>
<td>Defines blocks of code and automatically initializes arrays.</td>
</tr>
<tr>
<td>Brackets []</td>
<td>Declares array types and dereferences array values.</td>
</tr>
</tbody>
</table>
(b) Comments

Comments are descriptions that are added to a program to make code easier to understand. The compiler ignores comments and hence its only for documentation of the program.

**Java supports three comment styles.**

- **Block style comments** begin with /* and terminate with */ that spans multiple lines. Line style comments begin with // and terminate at the end of the line. (Shown in the above program)

- **Documentation style comments** begin with /** and terminate with */ that spans multiple lines. They are generally created using the automatic documentation generation tool, such as javadoc. (Shown in the above program) name of this compiled file is comprised of the name of the class with .class as an extension.

(c) White Space

White space consists mostly of the space character that you produce by hitting the space bar on your keyboard and that is commonly used to separate words in sentences. There are four other white space characters in Java, the horizontal tab, the form feed, the carriage return, and the linefeed.

2.2 Data Types

**Variables** are used for data that change during program execution. All variables have a name, a type, and a scope. The programmer assigns the names to variables, known as **identifiers**. An Identifier must be unique within a scope of the Java program. Variables have a **data type**, that indicates the kind of value they can store. Variables declared inside of a block or method are called local variables; They are not automatically initialized. The compiler will generate an error as a result of the attempt to access the local variables before a value has been assigned.

<table>
<thead>
<tr>
<th>Separators</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi colon ;</td>
<td>Terminates statements</td>
</tr>
<tr>
<td>Comma ,</td>
<td>Separates successive identifiers in variable declarations; chains statements in the test, expression of a for loop.</td>
</tr>
<tr>
<td>Period .</td>
<td>Selects a field or method from an object; separates Package names from sub-package and class names.</td>
</tr>
<tr>
<td>Used after loop labels.</td>
<td></td>
</tr>
</tbody>
</table>
The data type indicates the attributes of the variable, such as the range of values that can be stored and the operators that can be used to manipulate the variable. Java has four main primitive data types built into the language. You can also create your own composite data types.

Java has four main primitive data types built into the language. We can also create our own data types.

- **Integer**: Byte, short, int, and long.
- **Floating Point**: Float and double
- **Character**: Char
- **Boolean**: Variable with a value of true or false.

The following chart (Taken from Sun Java Site) summarizes the default values for the java built in data types. Since I thought Mentioning the size was not important as part of learning Java, I have not mentioned it in the below table. The size for each Java type can be obtained by a simple Google search.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Default Value (for fields)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>0</td>
<td>-127 to +128</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>-32768 to +32767</td>
</tr>
<tr>
<td>Int</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Long</td>
<td>0L</td>
<td></td>
</tr>
<tr>
<td>Float</td>
<td>0.0f</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>0.0d</td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td>‘\u0002</td>
<td>0 to 65535</td>
</tr>
<tr>
<td>String (object)</td>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>

When we declare a variable we assign it an identifier and a data type.

For Example

```java
String message = "hello world"
```

In the above statement, String is the **data type** for the **identifier** message. If you don’t specify a value when the variable is declared, it will be assigned the default value for its data type.
Identifier Naming Rules

- Can consist of upper and lower case letters, digits, dollar sign ($), and the underscore ( _ ) character.
- Must begin with a letter, dollar sign, or an underscore
- Are case sensitive
- Keywords cannot be used as identifiers
- Within a given section of your program or scope, each user defined item must have a unique identifier
- Can be of any length.

2.3 Operators in Java

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Increment/decrement operators
- Conditional Operator

2.3.1 The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Remainder after integer division</td>
</tr>
</tbody>
</table>
(a) Integer arithmetic

**Ex1:** Suppose a and b are integer variables and assigned a = 5 and b = 3 then.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Arithmetic Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>a+b</td>
<td>8</td>
</tr>
<tr>
<td>-</td>
<td>a-b</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>a*b</td>
<td>15</td>
</tr>
<tr>
<td>/</td>
<td>a/b</td>
<td>1</td>
</tr>
<tr>
<td>%</td>
<td>a%b</td>
<td>2</td>
</tr>
</tbody>
</table>

(b) Real arithmetic

Unlike C, java allows modulus operator to the float values. It returns float value after integer division.

Here quotient must be integer value.

**Ex2:** Suppose x and y are float variables and assigned x = 20.5 and y = 10.5 then.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Arithmetic Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>x+y</td>
<td>31.0</td>
</tr>
<tr>
<td>-</td>
<td>x-y</td>
<td>10.0</td>
</tr>
<tr>
<td>*</td>
<td>x*y</td>
<td>215.25</td>
</tr>
<tr>
<td>/</td>
<td>x/y</td>
<td>1.9523809523809523</td>
</tr>
<tr>
<td>%</td>
<td>x%y</td>
<td>10.0</td>
</tr>
</tbody>
</table>

(c) Mixed mode

When one of operands is real and the other is integer, the expression is called mixed-mode arithmetic expression. In an arithmetic expression all float operands must be converted as integer operands then the result will be in integer mode.

**Example**

(int) 12.0/5 gives result 2. Here 12.0 is float operand, 5 is an integer operand.
In the expression 12.0 converted into integer type as 12 and then divided by integer 5 gives result integer 2. In an arithmetic expression all integer operands must be converted as float operands then the result will be in float mode.

Example

12.0/(float) 5 gives result 2.4 Here 12.0 is float operand, 5 is an integer operand. In the expression 5 is converted into float type as 5.0 and then 12.0 is divided by float 5 gives result integer 2.4.

The program sbr.java for the demo of arithmetic operations

class sbr {
    public static void main(String args[]) {
        //a few numbers
        int i = 5;
        int j = 3;
        double x = 20.5;
        double y = 10.5;
        //adding numbers
        System.out.println("i=\"+i);
        System.out.println("j=\"+j);
        System.out.println("x=\"+x);
        System.out.println("y=\"+y);
        System.out.println("Adding");
        System.out.println(" i + j = "+(i + j));
        System.out.println(" x + y = "+(x + y));
        //subtracting numbers
        System.out.println("Subtracting");
        System.out.println(" i - j = "+(i - j));
        System.out.println(" x - y = "+(x - y));
    }
}
//multiplying numbers
System.out.println("Multiplying");
System.out.println(" i * j = " + (i * j));
System.out.println(" x * y = " + (x * y));

//dividing numbers
System.out.println("Dividing");
System.out.println(" i / j = " + (i / j));
System.out.println(" x / y = " + (x / y));

//computing the remainder resulting
//from dividing numbers
System.out.println("Modulus");
System.out.println(" i % j = " + (i % j));
System.out.println(" x % y = " + (x % y));

// Mixed mode arithmetic
System.out.println("12.0/(float)5 = " + 12.0/(float)5);
System.out.println("(int)12.0/5 = " +(int)12.0/5);

The output of the above program is as follows.
i=5
j=3
x=20.5
y = 10.5

Adding
  i + j = 8
  x + y = 31.0

Subtracting
  i - j = 2
  x - y = 10.0

Multiplying
  i * j = 15
  x * y = 215.25

Dividing
  i / j = 1
  x / y = 1.9523809523809523

Modulus
  i % j = 2
  x % y = 10.0

12.0 / (float) 5 = 2.4
(int) 12.0 / 5 = 2

2.3.2 The Relational Operators

There are six following relational operators supported by Java language:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Is Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Is Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Is Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Is Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Is Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Is Less than or equal to</td>
</tr>
</tbody>
</table>
Ex1: Suppose $a$, $b$ and $c$ are integer variables and assigned 3, 5 and 10 respectively.

<table>
<thead>
<tr>
<th>Relational Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a &gt; b$</td>
<td>False</td>
</tr>
<tr>
<td>$(a+b) &lt; c$</td>
<td>True</td>
</tr>
<tr>
<td>$a \geq 3$</td>
<td>True</td>
</tr>
<tr>
<td>$a \neq b$</td>
<td>True</td>
</tr>
<tr>
<td>$b \leq a$</td>
<td>False</td>
</tr>
<tr>
<td>$c == 10$</td>
<td>True</td>
</tr>
</tbody>
</table>

2.3.3 The Logical Operators

The following table lists the logical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>And</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Not</td>
</tr>
</tbody>
</table>

&& (and): If both operands are true then the logical and is true

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$a &amp;&amp; b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

||(or): If either operand is true then the logical or is true

| A | B | $a || b$ |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 1 | 0       |
| 1 | 0 | 0       |
| 1 | 1 | 1       |
! (not) : It negates the operand

<table>
<thead>
<tr>
<th>A</th>
<th>!a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Example: Suppose a, b and c are integer variables and assigned 3, 5 and 10 respectively.

<table>
<thead>
<tr>
<th>Logical Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a&gt;b) &amp;&amp; (c&gt;a)</td>
<td>False</td>
</tr>
<tr>
<td>(a&gt;b)</td>
<td></td>
</tr>
<tr>
<td>!(a&gt;8)</td>
<td>True</td>
</tr>
<tr>
<td>(b&gt;=a) &amp;&amp; (b&lt;=c)</td>
<td>True</td>
</tr>
</tbody>
</table>

2.3.4 Bitwise and Bit Shift Operators

In Java the bitwise and bit shift operators are used to manipulate the contents of variables at a bit level according to binary format. These operators perform bitwise and bit shift operations on integral type variables. There are different types of bitwise and bit shift operators available in the Java language summarized in the table.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name of the Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>Unary bitwise complement</td>
<td>~op2</td>
</tr>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
<td>op1 &amp; op2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bitwise inclusive OR</td>
</tr>
<tr>
<td>^</td>
<td>Bitwise exclusive OR</td>
<td>op1 ^ op2</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Signed left shift</td>
<td>op1 &lt;&lt; op2</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Signed right shift</td>
<td>op1 &gt;&gt; op2</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>Unsigned right shift</td>
<td>op1 &gt;&gt;&gt; op2</td>
</tr>
</tbody>
</table>

Let’s understand these operators in brief...
I. Unary Bitwise Complement (“~”)

The **unary bitwise complement** (“~”) operator takes a single bit and inverts it. In this case, the value of a bit which is 0 become 1 and vice versa. For example the value 7 to a variable “x” is represented in binary as 0111. But after applying “~” operator, the operation will be performed on each bit pattern which will return 1000 to the variable and the value 8 in the **decimal format**. Lets use the table to understand bitwise complement operation.

<table>
<thead>
<tr>
<th>A</th>
<th>~A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

II. Bitwise AND (\&)

The **Bitwise AND** (\&) operator performs the bitwise AND operation on each parallel pair of bits of two operands. The result is 1, if corresponding bits are 1 in both operands. Otherwise, the result is 0. Truth table follows.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp; B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

III. Bitwise inclusive OR (\|)

The **Bitwise inclusive OR** (\|) operator performs the bitwise inclusive OR operation on each parallel pair of bits of two operands. In each pair, the result is 1, if either first or second bit is 1 (or both are 1). Otherwise the result is 0. Truth table follows.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
IV. Bitwise exclusive OR (^)

The Bitwise exclusive OR (^) performs the exclusive or (XOR) operation i.e. The result in each position is 1 if the two bits are different, and 0 if they are the same. Truth table follows.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ^ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bit Shifts Operators

The computer processor has the registers including a fixed number of available bits for storing numerals. So it is possible to “shift out” some bits of the register at one end, and “shift in” from the other end. The number of bits are shifted within the range mode of 32.

The bit shifts operators are used to perform bitwise operations on the binary representation of an integer instead of its numerical value. In this operation, the bit shifts operators don’t operate the pairs of corresponding bits rather the digits are moved, or shifted in a computer register either to the left or right according to the distance specified by a number.

Sign Bit

A sign bit is found in the left most position of the number and is know as most significant bit (MSB) which indicates the status of a number i.e. the number is positive or negative. If the value of the sign bit is 0 then the number is positive; otherwise the number is negative, if the value of the sign bit is 1.

Now lets understand these operators in brief.

I. Signed Left Shift (“<<“)

The signed left shift (“<<“) operator shifts a bit or bits to the left by the distance specified in the right operand. In this case, the leftmost digit is shifted at the end of the register, and a new 0 is shifted into the rightmost position. No matter, the number is positive or negative; In both of case the leading bit position is always filled with a zero.
This diagram shows that, all bits of the upper position were shifted to the left by the distance of 1; and the Zero was shifted to the right most position. Thus the result is returned as 11100.

Another expression “2<<2”; shifts all bits of the number 2 to the left placing a zero to the right for each blank place. Thus the value 0010 becomes 1000 or 8 in decimal.

II. Signed Right Shift (“>>”)

The signed right shift (“>>”) operator shifts a bit (or bits) to the right by the distance specified in the right operand and fills the left most bit by the sign bit. In this case the rightmost bit (or bits) is shifted out, and a new 0 is filled with the sign bit into the high-order bits to the left position if the left operand is positive; otherwise 1, if the left operand is negative. This technique is known as sign extension.

This diagram shows that, all bits of the upper position were shifted to the right distance specified by 1. Since the sign bit of this number indicates it as a positive number so the 0 is shifted to the right most position. Thus the result is returned as 00011 or 3 in decimal.
Another expression “2>>2”; shifts all bits of the number 2 to the right placing a zero to the left for each blank place. Thus the value 0010 becomes 0000 or 0 in decimal.

When signed left or signed right shifting operation is performed then the sign bit is ignored i.e. all the bits except the sign bit can be moved but the sign bit stays the same. Thus a signed left or signed right shift (<< and >>) operator never causes a number to change its sign. A positive number will always stay positive and a negative number will always stay negative. But the result for a negative number is different.

For example, if we take a negative number as -50 then this value is represented in binary as 11001110 then the expression “-50>>2”; will return the result as 11110011 or -13 in decimal.

III. Unsigned Right Shift (“>>>”)

The unsigned right shift (“>>>”) operator behave like the signed right shift operator. i.e. it shifts a bit (or bits) to the right. But unlike “>>” operator, this operator always shifts zeros into the leftmost position by the distance specified in the right operand. So the result of applying the >>> operator is always positive.

For example, the expression “14>>>2”; shifts all bits of the number 14 to the right placing a zero to the left for each blank place Thus the value 1110 becomes 0011 or 3 in decimal.

An unsigned shift operation of a negative number generally returns the result in a positive number, because any unsigned right shift operation replaces the leading sign bit with a zero which indicates a positive number.

The following program BitwiseOperators.java demonstrates the bitwise operators.

class BitwiseOperators {
    public static void main(String args[])
    {
        int a = 7; // 0111
        int b = 13; // 1101
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("a & b : +(a & b));
    
}
System.out.println("a | b : +(a | b));
System.out.println("a ^ b : +(a ^ b));
System.out.println("~a : +(~a));
System.out.println("a << 2 : +(a << 2));
System.out.println("a >> 2 : +(a >> 2));
System.out.println("a >>> 1 : +(a >>> 1));
}
}

Output of the Program

a = 7
b = 13
a & b : 5
a | b : 15
a ^ b : 10
~a : -8
a << 2 : 28
a >> 2 : 1
a >>> 1 : 3

2.3.5 The Assignment Operators

There are following assignment operators supported by Java language

<table>
<thead>
<tr>
<th>Operator</th>
<th>Expression</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x+=a</td>
<td>x=x+a</td>
</tr>
<tr>
<td>-=</td>
<td>x-=a</td>
<td>x=x-a</td>
</tr>
<tr>
<td>*=</td>
<td>x*=a</td>
<td>x=x*a</td>
</tr>
<tr>
<td>/=</td>
<td>x/=a</td>
<td>x=x/a</td>
</tr>
<tr>
<td>%=</td>
<td>x%=a</td>
<td>x=x %a</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
<td>x&lt;&lt;=2</td>
<td>x=x &lt;&lt; 2</td>
</tr>
</tbody>
</table>
There are two Increment or decrement operators -- ++ and --. These two operators are unique in that they can be written both before the operand they are applied to, called prefix increment/decrement(++) or after, called postfix increment/decrement(--). The meaning is different in each case.

**Example**

```java
x = 5;
y = ++x;
System.out.println(y);
prints 6, but
x = 5;
y = x++;
System.out.println(y);
prints 5
When we write i++ we’re using shorthand for i = i + 1.
When we say i-- we’re using shorthand for i = i - 1.
```

**Example**

```java
x = 5;
y = --x;
System.out.println(y);
prints 4, but
x = 5;
```
y = x--; 
System.out.println(y);
prints 5

2.3.7 Conditional Operator ( ? : )

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate boolean expressions. The goal of the operator is to decide which value should be assigned to the variable.

The operator is written as

variable x = (Boolean expression)? Expr1 : expr2

Boolean Expression is evaluated first and if it is true expression1 is assigned to the variable x.

if it is false expression2 is assigned to the variable x.

Following is the example

```java
public class Test {   public static void main(String args[]){      int a , b;      a = 10;      b = (a == 1) ? 20 : 30;      System.out.println( "Value of b is : " + b );      b = (a == 10) ? 20: 30;      System.out.println( "Value of b is : " + b );   }}
```

This would produce following result

Value of b is : 30Value of b is : 20

Precedence of Java Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others;

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>X= 5 + 3 *2</td>
<td>X=11</td>
</tr>
<tr>
<td>X=(5+3) *2</td>
<td>X=16</td>
</tr>
<tr>
<td>X=8+4/(3-1)</td>
<td>X=10</td>
</tr>
<tr>
<td>X=8/4/2</td>
<td>X=1</td>
</tr>
</tbody>
</table>

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.
2.4.1 Creating a Method

In general, a method has the following syntax

    modifier returnType methodName(list of parameters) { // Method body;

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, overload methods using the same names, and apply method abstraction in the program design.

### Table of Operators

<table>
<thead>
<tr>
<th>Category</th>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postfix</td>
<td>() [] . (dot operator)</td>
<td>Left to right</td>
</tr>
<tr>
<td>Unary</td>
<td>++ - - ! ~</td>
<td>Right to left</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>* / %</td>
<td>Left to right</td>
</tr>
<tr>
<td>Additive</td>
<td>+ -</td>
<td>Left to right</td>
</tr>
<tr>
<td>Shift</td>
<td>&gt;&gt; &gt;&gt;&gt; &lt;&lt;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Relational</td>
<td>&gt; &gt;= &lt; &lt;=</td>
<td>Left to right</td>
</tr>
<tr>
<td>Equality</td>
<td>== !=</td>
<td>Left to right</td>
</tr>
<tr>
<td>Bitwise AND</td>
<td>&amp; ! =</td>
<td>Left to right</td>
</tr>
<tr>
<td>Bitwise XOR</td>
<td>^</td>
<td>Left to right</td>
</tr>
<tr>
<td>Bitwise OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical AND</td>
<td>&amp;&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Logical OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional</td>
<td>?:</td>
<td>Right to left</td>
</tr>
<tr>
<td>Assignment</td>
<td>+= -= *= /= %=</td>
<td>Right to left</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;= &gt;&gt;= &amp;= ^=</td>
<td>=</td>
</tr>
<tr>
<td>Comma</td>
<td>,</td>
<td>Left to right</td>
</tr>
</tbody>
</table>
A method definition consists of a method header and a method body. Here are all the parts of a method:

**Modifiers**: The modifier, which is optional, tells the compiler how to call the method. This defines the access type of the method.

**Return Type**: A method may return a value. The `return ValueType` is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the `return ValueType` is the keyword `void`.

**Method Name**: This is the actual name of the method. The method name and the parameter list together constitute the method signature.

**Parameters**: A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

**Method Body**: The method body contains a collection of statements that define what the method does.

```
/** Return the max between two numbers */
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

**Note**: In certain other languages, methods are referred to as procedures and functions. A method with a nonvoid return value type is called a function; a method with a void return value type is called a procedure.

**Example**

Here is the source code of the above defined method called `max()`. This method takes two parameters `num1` and `num2` and returns the maximum between the two.

```java
/** Return the max between two numbers */
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```
result;   if (num1 > num2)      result = num1;
else      result = num2;   return result; }

2.4.2 Calling a Method

In creating a method, you give a definition of what the method is to do. To use a method, you have to call or invoke it. There are two ways to call a method; the choice is based on whether the method returns a value or not.

When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method-ending closing brace is reached.

If the method returns a value, a call to the method is usually treated as a value.

For example

    int larger = max(30, 40);

If the method returns void, a call to the method must be a statement. For example, the method println returns void. The following call is a statement:

    System.out.println("Welcome to Java!");

Example

Following is the example to demonstrate how to define a method and how to call it.

    public class TestMax {   /** Main method */
    public static void main(String[] args) {      int
i = 5;      int j = 2;      int k = max(i, j);      System.out.println("The maximum between "+ i + " and "+ j + " is "+ k); }/** Return the max between two numbers */
    public static int max(int num1, int num2) {   int result;   if (num1 > num2)      result = num1;
else      result = num2;   return result; }

This would produce following result

    The maximum between 5 and 2 is 5

This program contains the main method and the max method. The main method is just like any other method except that it is invoked by the JVM.

The main method’s header is always the same, like the one in this example, with the modifiers public and static, return value type void, method name main,
and a parameter of the String[] type. String[] indicates that the parameter is an array of String.

2.4.3 The void Keyword

This section shows how to declare and invoke a void method. Following example gives a program that declares a method named printGrade and invokes it to print the grade for a given score.

Example

```java
public class TestVoidMethod {
    public static void main(String[] args) {
        printGrade(78.5);
    }
    public static void printGrade(double score) {
        if (score >= 90.0) {
            System.out.println('A');
        } else if (score >= 80.0) {
            System.out.println('B');
        } else if (score >= 70.0) {
            System.out.println('C');
        } else if (score >= 60.0) {
            System.out.println('D');
        } else {
            System.out.println('F');
        }
    }
}
```

This would produce following result

C

Here the printGrade method is a void method. It does not return any value. A call to a void method must be a statement. So, it is invoked as a statement in line 3 in the main method. This statement is like any Java statement terminated with a semicolon.

2.5 Type conversion and casting

2.5.1 Type conversions in expressions

Java permits mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversion. We know that the computer considers one operator at a time, involving two operands. If the operands are of different, the ‘lower’ type is automatically converted to the ‘higher’ type before the operation proceeds. The result is of the higher type.

If byte, short and int variables are used in an expression, the result is always promoted to int, to avoid overflow. If an expression contains a float operand, the entire expression is promoted to float. If any operand is double, result is double.
The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.

1. Float to int causes truncation of the fractional part.
2. Double to float causes rounding of digits.
3. Long to int causes dropping of the excess higher order bits.

### 2.5.2 Casting a value

We have already discussed how Java performs type conversion automatically. However, there are instances when we want to force a type conversion in a way that is different from the automatic conversion. Consider, for example, the cost of apples is

\[
\text{cost} = \frac{\text{no of apples}}{\text{amount}}
\]

Since no of apples and amount are declared as integers in the program, the decimal part of the result of the division would be lost and ratio would not represent a correct figure. This problem can be solved by converting locally one of the variables to the floating point as shown below:

\[
\text{cost} = \left(\text{float}\right) \frac{\text{no of apples}}{\text{amount}}
\]

The operator (float) converts the no of apples to floating point for the purpose of evaluation of the expression. Then using the rule of automatic conversion, the division is performed in floating point mode, thus retaining the fractional part of result.

### Automatic Type Conversion Chart

<table>
<thead>
<tr>
<th></th>
<th>Char</th>
<th>Byte</th>
<th>Short</th>
<th>Int</th>
<th>Long</th>
<th>Float</th>
<th>Double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>long</td>
<td>Float</td>
<td>Double</td>
</tr>
<tr>
<td>Byte</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>long</td>
<td>Float</td>
<td>Double</td>
</tr>
<tr>
<td>Short</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>long</td>
<td>Float</td>
<td>Double</td>
</tr>
<tr>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>Int</td>
<td>long</td>
<td>Float</td>
<td>Double</td>
</tr>
<tr>
<td>Long</td>
<td>long</td>
<td>Long</td>
<td>long</td>
<td>Long</td>
<td>long</td>
<td>Float</td>
<td>Double</td>
</tr>
<tr>
<td>Float</td>
<td>Float</td>
<td>Float</td>
<td>Float</td>
<td>Float</td>
<td>Float</td>
<td>Float</td>
<td>Double</td>
</tr>
<tr>
<td>Double</td>
<td>Double</td>
<td>Double</td>
<td>Double</td>
<td>Double</td>
<td>Double</td>
<td>Double</td>
<td>Double</td>
</tr>
</tbody>
</table>
Note that in no ways does the operator (float) affect the value of the variable no_of_apples. And also, the type of no_of_apples remains as int in the other parts of the program.

The process of such a local conversion is known as casting a value. The general form of a cast is:

(Type_name) expression

where type-name is one of the standard data types. The expression may be a constant, variable or an expression. Some examples of casts and their actions are shown in table 2.1.

Use of casts

<table>
<thead>
<tr>
<th>Example</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>x= (int) 8.4</td>
<td>8.4 is converted to integer as 8 by truncation</td>
</tr>
<tr>
<td>a= (int) 21.3 / (int) 4.5</td>
<td>Evaluated as 21/4 and the result would be 5</td>
</tr>
<tr>
<td>y=(int) (a+b)</td>
<td>The result of a+b is converted to integer</td>
</tr>
<tr>
<td>z=(int) a + b</td>
<td>A is converted to integer and then added to b</td>
</tr>
</tbody>
</table>

2.6 Constructors and destructors

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

Example

Here is a simple example that uses a constructor

```java
    // A simple constructor.class MyClass {   int
    x;   // Following is the constructor  MyClass()   
{      x = 10;   }
```
You would call constructor to initialize objects as follows

```
class ConsDemo {   public static void
main(String args[]) {      MyClass t1 = new
MyClass();      MyClass t2 = new MyClass();
System.out.println(t1.x + " " + t2.x);   }}
```

This would produce following result

```
10 10
```

Most often you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method: just declare them inside the parentheses after the constructor’s name.

**Example**

Here is a simple example that uses a constructor

```
// A simple constructor.class MyClass {   int x;      // Following is the constructor   MyClass(int i ) {      x = i;   }}
```

You would call constructor to initialize objects as follows

```
class ConsDemo {   public static void
main(String args[]) {      MyClass t1 = new
MyClass( 10 );      MyClass t2 = new MyClass( 20 );
System.out.println(t1.x + " " + t2.x);   }}
```

This would produce following result

```
10 20
```

### 2.6.1 Destructors

**The finalize() Method**

It is possible to define a method that will be called just before an object’s final destruction by the garbage collector. This method is called `finalize()`, and it can be used to ensure that an object terminates cleanly.

For example, you might use `finalize()` to make sure that an open file owned by that object is closed. To add a finalizer to a class, you simply define the `finalize()` method. The Java runtime calls that method whenever it is about to recycle an object of that class.
Inside the finalize( ) method you will specify those actions that must be performed before an object is destroyed.

**The finalize( ) method has this general form**

```java
protected void finalize( ){   // finalization code here}
```

Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class.

This means that you cannot know when or even if finalize( ) will be executed. For example, if your program ends before garbage collection occurs, finalize( ) will not execute.

**Summary**

We have discussed all the basic data types and operators available in Java and also seen their use in the expressions. Type conversion and order of precedence of operators during the evaluation of expressions have been highlighted.

**Short Answer Type Questions**

1. What is a class?
2. What are keywords?
3. What is an identifier?
4. What is a literal?
5. What are comments?
6. What are four main primitive data types?
7. What are white space characters?
8. What is mixed mode?
9. Write the difference between ++i and i++
10. Write the difference between --i and i--
11. What is a conditional operator?
12. What is a method?
13. What is a constructor?
14. What is a destructor?
Long Answer Type Questions

1. Write about separators in java.
2. Explain the Arithmetic operators with examples.
3. Explain the Relational operators with examples.
4. Explain the Logical operators with examples.
5. Explain the Bitwise and Bitshift operators with examples.
6. Explain the assignment operators with examples.
7. Explain the increment/decrement operators with examples.
8. Write about precedence of operators.
9. Write a java method to find minimum value in given two values.
**Learning Objectives**

After studying this unit, the student will be able to understand

- If statements and switch statement
- Loops (while, do..while & for)
- Break and continue statements
- Single and two dimensional arrays

**3.0 Introduction**

A Java program is a set of statements, which are normally executed sequentially in the order in which they appear. However in general programs, we have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are met. This causes decision making and loops.
When a program breaks the sequential flow and jumps to another part of the code, it is called branching. When the branching is based on a particular condition, it is called conditional branching. If branching takes place without any condition, it is called unconditional branching.

### 3.1 Decision making

There are two types of decision making (Branching) statements in Java.

**They are**
- If statements
- Switch statements

#### 3.1.1 The if Statement

An if statement consists of a Boolean expression followed by one or more statements.

**Syntax**

The syntax of an if statement is

```java
if (Boolean_expression)
{
    // Statements will execute if the Boolean expression is true
}
```

If the boolean expression evaluates to true then the block of code inside the if statement will be executed. If it is false simply bypass the block of code.

```java
// finds larger value between two values
import java.util.Scanner;
class larger
{
    public static void main(String args[])
    {
        int x, y, big;
        System.out.println("Enter two integers ");
        Scanner in = new Scanner(System.in);
```
```java
x = in.nextInt();
y = in.nextInt();
    big = x;
    if(big<y) big = y;
    System.out.println("big="+big);
}
}

Output 1

Enter two integers
3  6
big=6
output2
Enter two integers
12  5
big=12
// finds largest value among three values
import java.util.Scanner;
class largest
{
    public static void main(String args[])
    {
        int x, y, z,big;
        System.out.println("Enter three integers ");
        Scanner in = new Scanner(System.in);
        x = in.nextInt();
y = in.nextInt();
z = in.nextInt();
```
big = x;
if(big<y) big = y;
if(big<z) big = z;
System.out.println("biggest=":"+big);
}
}

Output

Enter three integers
12 34 5
biggest =34

Example

public class Test {
public static void main(String args[]){
    int x = 10;

    if( x < 20 ){
        System.out.print("This is if statement");
    }
}

This would produce following result
This is if statement

3.1.2 The if...else Statement

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

Syntax

If (Boolean expression) expr1; else expr2;
If Boolean expression is true then the expr1 is executed. If Boolean expression is false expr2 is executed.

**Example**

```java
// finds smaller value between two values
import java.util.Scanner;

class smaller
{
    public static void main(String args[])
    {
        int x, y ,small;
        System.out.println("Enter two integers ");
        Scanner in = new Scanner(System.in);
        x = in.nextInt();
        y = in.nextInt();

        if (x<y) small = x; else small = y;
        System.out.println("small=",small);
    }
}
```

**Output**

Enter two integers
3  6
small=6

**Example 2**

```java
public class Test {
    public static void main(String args[]){
        int x = 30;
```
if( x < 20 ){
    System.out.print(“This is if statement”);
} else {
    System.out.print(“This is else statement”);
}
}

This would produce following result

This is else statement

3.1.3 The if...else if...else Statement

An if statement can be followed by an optional else if...else if statement, which is very useful to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind.

• An if can have zero or one else’s and it must come after any else if’s.
• An if can have zero to many else if’s and they must come before the else.
• Once an else if succeeds, none of the remaining else if’s or else’s will be tested.

Syntax

The syntax of a if...else is

if(Boolean_expression 1){
    //Executes when the Boolean expression 1 is true
} else if(Boolean_expression 2){
    //Executes when the Boolean expression 2 is true
} else if(Boolean_expression 3){
    //Executes when the Boolean expression 3 is true
} else {

//Executes when the none of the above condition is true.

}

Example

public class Test {
    public static void main(String args[]){
        int x = 30;

        if( x == 10 ){
            System.out.print("Value of X is 10");
        }else if( x == 20 ){
            System.out.print("Value of X is 20");
        }else if( x == 30 ){
            System.out.print("Value of X is 30");
        }else{
            System.out.print("This is else statement");
        }
    }
}

This would produce following result

Value of X is 30

3.1.4 Nested if...else Statement

It is always legal to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement.

Syntax

The syntax for a nested if...else is as follow

if(Boolean_expression 1){
    //Executes when the Boolean expression 1 is true
    if(Boolean_expression 2){
}
//Executes when the Boolean expression 2 is true
}
}
You can nest else if...else in the similar way as we have nested if statement.

**Example**

```java
class Test {
    public static void main(String args[]){
        int x = 30;
        int y = 10;

        if( x == 30 ){
            if( y == 10 ){
                System.out.print("X = 30 and Y = 10");
            }
        }
    }
}
```

This would produce following result

X = 30 and Y = 10

**3.1.5 The switch Statement**

A switch statement is used instead of multiple if statements. A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

**Syntax**

The syntax of enhanced for loop is

```java
switch(expression){
    case value :
        //Statements
    //Statements
}```
The following rules apply to a switch statement

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Example

    public class Test {
        public static void main(String args[]) {
            char grade = args[0].charAt(0);
switch(grade)
{
    case 'A':
        System.out.println("Excellent!");
        break;
    case 'B':
    case 'C':
        System.out.println("Well done");
        break;
    case 'D':
        System.out.println("You passed");
    case 'F':
        System.out.println("Better try again");
        break;
    default:
        System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
}

Compile and run above program using various command line arguments. This would produce following result.
D : / java > java Test A
Excellent ;
Your grade is A
D : / java > java Test B
Well done
3.2 Loops

There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop.

Java has very flexible three looping mechanisms. You can use one of the following three loops.

• While Loop
• Do...while Loop
• For Loop

3.2.1 The while Loop

A while loop is a control structure that allows you to repeat a task a certain number of times.

Syntax

The syntax of a while loop is

```java
while(Boolean_expression)
{
    //Statements
}
```
When executing, if the boolean_expression result is true then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Here key point of the while loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example 1

// prints 5 natural numbers 1, 2, 3, 4 and 5
class numbers
{
    public static void main(String args[]) {
        int i;

        i = 1;
        while (i <= 5) {
            System.out.println(i++);
        }
    }
}

Output

1
2
3
4
5

Example 2

// prints odd numbers 1, 3, 5, 7 and 9
class numbers
{
    public static void main(String args[]) {
        int i;

        i = 1;
        while (i <= 5) {
            System.out.println(i++);
        }
    }
}
i=1;
while(i<=9) {
    System.out.println(i);
    i=i+2;
} }

Output
1
3
5
7
9

Example 3
// prints 5 natural numbers decreasing order 5,4,3,2 and 1
class numbers {
    public static void main(String args[]) {
        int i;

        i=5;
        while(i>=1) {
            System.out.println(i--);
        }
    }
}

Output
5
4
3
Example 4

// finds factorial by using while loop
import java.util.Scanner;

class factorialw
{
    public static void main(String Args[])
    {
        int i,f=1,n;
        System.out.println("enter an integer");
        Scanner in = new Scanner(System.in);
        n = in.nextInt();
        i=1;
        while(i<=n){
            f*=i; ++i;
        }
        System.out.println("factorial of " +n+ " is " +f);
    }
}

Output

    enter an integer
    5
    factorial of 5 is 120.

3.2.2 The do...while Loop

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

The syntax of a do...while loop is

Do
Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example

// prints 5 natural numbers 1,2,3,4 and 5 by using do while
class numbers
{
    public static void main(String args[])
    {
        int i;

        i=1;
        do {
            System.out.println(i++);
        } while(i<=5);
    }
}

This would produce following result

1
2
3
4
5

Example

// finds sum of n natural numbers by using do while
import java.util.Scanner;

class SumDoWhile
{
public static void main(String Args[])
{
int i,s=0,n;
System.out.println("enter an integer");
Scanner in = new Scanner(System.in);
n =  in.nextInt();
i=1;
do {
  s+=i; ++i;}
  while(i<=n);
System.out.println("sum of "+n+ " natural numbers is " +s);
}
}

enter an integer
5
sum of 5 natural numbers is 15

3.2.3 The for Loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

Syntax

The syntax of a for loop is

For(expr1;expr2;expr3){statements}

Expr 1: initialization

Expr 2
Expr 3

for(initialization; Boolean_expression; update)
{
    //Statements
}

Here is the flow of control in a for loop

1. The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

2. Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement of the for loop.

3. After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.

4. The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

Example

```java
public class Test {
    public static void main(String args[]){
        for(int x = 1; x < 5; x = x+1){
            System.out.print(x +
            System.out.print("\n");
        }
    }
}
```
This would produce following result

1

2

3

4

5

3.3 Break and continue

3.3.1 The break Keyword

The break keyword is used exit from a loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax

The syntax of a break is a single statement inside any loop

break;

Example

public class Test {
    public static void main(String args[]){
        int[] numbers = {10, 20, 30, 40, 50};
        for(int x : numbers ){
            if( x == 30 ){
                break;
            }
        }
        System.out.println( x );
    }
}

This would produce following result

10
3.3.2 The continue Keyword

The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax

The syntax of a continue is a single statement inside any loop

```
continue;
```

Example

```
public class Test {
    public static void main(String args[]) {
        int[] numbers = {10, 20, 30, 40, 50};

        for (int x : numbers) {
            if (x == 30) {
                continue;
            }
            System.out.print(x);
            System.out.print("n");
        }
    }
}
```

This would produce following result

```
10
20
```
3.4 Arrays

3.4.1 Single dimensional Arrays

Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type.

\[
\begin{align*}
A[0] &= 12 \\
\end{align*}
\]

In the above A is the array name, 0, 1, 2, 3, and 4 are indexes, 12, -5, 18, 2, and 16 are data elements. Array A containing 5 integer type elements.

An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Declaration of Arrays

Arrays in java may be declared in two forms

type arrayname[];

Form1 type arrayname[];
Form2 type[] arrayname;

Example

```java
int number[];
float average[];
int[] counter;
float[] marks;
```

Remember, we do not give the size of the array in the declaration.
Creation of Arrays

You can create an array by using the new operator with the following syntax

Arrayname = new datatype[array size]

Example

number = new int[5];
average = new float[10];

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below.

datatype[] arrayname = new datatype[arraySize];

int[] a = new int[5];

Alternatively you can create arrays as follows

datatype[] arrayname = {value0, value1, ..., valuek};

The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayname.length - 1.

Example

int a[] = {12, -5,18,2,16 }

Processing Arrays

When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.

Example

Here is a complete example of showing how to create, initialize and process arrays.

public class TestArray {
    public static void main(String[] args) {
        int[] a = {5,2,-3,8,6};

        // Print all the array elements
System.out.println("Given Array");
for (int i = 0; i < a.length; i++) {
    System.out.println(a[i]);
}
// Summing all elements
int total = 0;
for (int i = 0; i < a.length; i++) {
    total += a[i];
}
System.out.println("Total is " + total);
// Finding the largest element
int max = a[0];
for (int i = 1; i < a.length; i++) {
    if (a[i] > max) max = a[i];
}
System.out.println("Maximum value is " + max);
}

The output of the above program is as follows

**Given Array**

5
2
-3
8
6

Total is 18
Maximum value is 8
3.4.2 Two-dimensional arrays

Two-dimensional arrays are defined as “an array of arrays”. Since an array type is a first-class Java type, we can have an array of ints, an array of Strings, or an array of Objects. For example, an array of ints will have the type int[]. Similarly we can have int[][]], which represents an “array of arrays of ints”. Such an array is said to be a two-dimensional array. It has rows and columns as follows.

The command

```java
int[][] a = new int[3][4]
```

<table>
<thead>
<tr>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rows</th>
<th>0</th>
<th>12</th>
<th>5</th>
<th>-6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>25</td>
<td>4</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

- \(a[0][0] = 12\)
- \(a[0][1] = 5\)
- \(a[0][2] = -6\)
- \(a[0][3] = 8\)
- \(a[1][0] = 9\)
- \(a[1][1] = 4\)
- \(a[1][2] = 0\)
- \(a[1][3] = 7\)
- \(a[2][0] = 13\)
- \(a[2][1] = 25\)
- \(a[2][2] = 4\)
- \(a[2][3] = 14\)

In the above array named \(a\) contains 3 rows and 4 columns.

To process a two-dimensional array, we use nested for loops. We already know about for loop.
A loop in another loop is called a Nested loop.

The following program AddTwoMatrix.java is for matrix addition

```java
// Matrix Addition
import java.util.Scanner;
class AddTwoMatrix {
    public static void main(String args[])
    {
        int m, n, c, d;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of matrix");
        m = in.nextInt();
        n = in.nextInt();

        int first[][] = new int[m][n];
        int second[][] = new int[m][n];
        int sum[][] = new int[m][n];
        System.out.println("Enter the elements of first matrix");

        for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
                first[c][d] = in.nextInt();
        System.out.println("Enter the elements of second matrix");

        for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
                second[c][d] = in.nextInt();

        for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
                sum[c][d] = first[c][d] + second[c][d];

        System.out.println("Sum of the matrices:");

        for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
                System.out.print(sum[c][d] + " ");
    }
}
```

second[c][d] = in.nextInt();
for ( c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
        sum[c][d] = first[c][d] + second[c][d];
System.out.println("First Matrix :-");
for ( c = 0 ; c < m ; c++ )
{
    for ( d = 0 ; d < n ; d++ )
        System.out.print(first[c][d] + "\t");
    System.out.println();
}
System.out.println("Second Matrix:-");
for ( c = 0 ; c < m ; c++ )
{
    for ( d = 0 ; d < n ; d++ )
        System.out.print(second[c][d] + "\t");
    System.out.println();
}
System.out.println("Sum of entered matrices:-");
for ( c = 0 ; c < m ; c++ )
{
    for ( d = 0 ; d < n ; d++ )
        System.out.print(sum[c][d] + "\t");
    System.out.println();
}
The output of the program is

Enter the number of rows and columns of matrix

2
2

Enter the elements of first matrix

1 2
3 4

Enter the elements of second matrix

5 6
7 8

First Matrix

1 2
3 4

Second Matrix

5 6
7 8

Sum of entered matrices

6 8
10 12

Summary

The decision making and looping are very extremely important tools in developing programs. Whenever the computer runs a Java program, it goes straight from the first line of code to the last. Control statements allow you to change the computer’s control from automatically reading the next line of code to reading a different one.

Control statements are used in programming languages to cause the flow of control to advance and branch based on changes to the state of a program.

Control Statements in Java are very simple. In some languages they have complex control statements, that do a lot. However, Java offers only simple
control statements. If you need a complex control statement, you can build it with a few simple ones. This makes control statements a lot easier to work with. Although Java has a few different kinds, I will illustrate one more common one. The loop. A loop is where you follow the same code over and over again, until some condition stops it.

The if statement executes a block of code only if the specified expression is true. If the value is false, then the if block is skipped and execution continues with the rest of the program. You can either have a single statement or a block of code within an if statement. Note that the conditional expression must be a Boolean expression.

**Switch Case Statement** The switch case statement, also called a case statement is a multi-way branch with several choices. A switch is easier to implement than a series of if/else statements.

The examples discussed here are simple but this concept will be certainly most useful in developing complex systems.

**Short Answer Type Questions**

1. What is conditional branching ?
2. What is unconditional branching ?
3. Write the syntax of if statement.
4. Write a program to find bigger value in given two values.
5. What is switch statement ?
6. What are nested if statements ?
7. What are various loop statements ?
8. What is a break statement ?
9. What is a continue statement ?
10. What is an array ?

**Long Answer Type Questions**

1. Write about if and if..else statements.
2. Write about switch statement?
3. Write a program to find biggest value among three values.
4. Write about while loop with an example.
5. Write about do-while loop with an example.
6. Write about for loop with an example.
7. Write a program to find factorial of a given number.
8. Write a program to find natural sum of a given number.
9. Write a program to find minimum value from a given array.
10. Write a program to find sum of values of an array.
11. Write a program for matrix addition.
12. Write a program for matrix multiplication.
Implementation of OOP in Java

Structure

4.1 Polymorphism
4.2 Inheritance
4.3 Overloading

Learning Objectives

After studying this unit, the student will be able to understand

- Implementation of Polymorphism, Inheritance, Overloading

4.1 Polymorphism

Poly = many, morphism = phases.

We can store all the objects of extended classes in to variable of parent class. It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.
The following BoxDemo.java program explains polymorphism

class Box {
    int w,h;
    void info() {
        System.out.println("This is a simple box");
        System.out.println("width = " + w + " height " + h);
    }
}
class woodenBox extends Box {
    int life;
    void info() {
        System.out.println("This is a wooden box");
    }
}
class SteelBox extends Box {
    int wg;
    void info() {
        System.out.println("This is a steel box");
    }
}
class LargewoodenBox extends woodenBox {
    void info() {
        System.out.println("This is a Huge wooden box");
    }
}
class BoxDemo {
    public static void main ( String ary[] ) {
        Box x;
        Box b1 = new Box();
        woodenBox wb = new woodenBox();
        SteelBox s1 = new SteelBox();
LargewoodenBox p1=new LargewoodenBox();
    b1.info();
    wb.info();
    s1.info();
    p1.info();  }  }

The output of the above program

Fig 4.1

4.2 Inheritance

Inheritance is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance is that you can create new classes that are built upon existing Classes. When you inherit from an existing class, you reuse(or inherit) methods and fields, and you add new methods and fields to adapt your new class to new situations. Inheritance represents the IS-A Relationship.

Syntax of inheritance

```java
class <subclass-name> extends <superclass-name>
{
    // methods and fields
}
```
The keyword **extends** indicates that you are making a new class that derives from an existing class. In the terminology of java, a class that is inherited is called a superclass. The new class is called a subclass.

As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is programmer IS-A employee. It means programmer is a type of employee.

The InheritanceExample.java shows that how the values are displaying through inheritance.

```java
class Employee{
    int salary = 60000;
}

class Programmer extends Employee{
    int bonus = 10000;
}

class InterfaceExample{
    public static void main(String args[])
    {
        Programmer p = new Programmer();
        System.out.println("programmer salary is:" + p.salary);
        System.out.println("programmer bonus is:" + p.bonus);
    }
}
```
4.3 Overloading

Overloading means creating more than a single method with same name with different signatures. In the example three methods are created with same name. Java understands these methods with there signatures. Java identifies the methods by comparing their signatures like return types, constructor parameters & access modifier used.

Method Overloading.java explains the overloading

class Overload {
    void test(int a) {
        System.out.println("a: + a");
    }
    void test(int a, int b) {
        System.out.println("a and b: + a + " + b);
    }
    double test(double a) {
        System.out.println("double a: + a");
        return a*a;
    }
}

class MethodOverloading {
    public static void main(String args[]) {
        Overload overload = new Overload();
        double result;
overload.test(10);
overload.test(20, 30);
result = overload.test(2.5);
System.out.println("Result : " + result);

Output of the above program

![Fig 4.3]

Summary

Java is an object oriented language. It enables us not only to organize our program code into logical units called objects but also to take advantage of polymorphism, inheritance and overloading.

Long Answer Type Questions

1. Explain the polymorphism with an example.
2. Explain the Inheritance with an example.
3. Explain the overloading with an example.
5.1 Concepts and Uses of Packages

Java is a package-centric language; we can simply assume that for good organization and name scoping, we would put all our classes into packages.

Ex: Some programmers in a company working different parts of a project, write a class named “utilities”. If all the persons are not declared a class with name “utilities” in any explicit package, and are in the classpath, we won’t have any way to tell the compiler or JVM which of all we are trying to reference. Packages works like a “class libraries” in other languages. Nothing but achieving the concept of “reusability”.

5.2 Creating and importing Packages

5.3 Concepts of interfaces and uses

5.4 Writing and implementing simple interfaces

Learning Objectives

After studying this unit, the student will be able to

- What is a package
- What is an interface
- Differences between interfaces and packages
The first statement java program is a package statement. This statement declares a package name and informs the compiler that the classes defined here belong to this package.

Ex: package student

But, the package statement is in a java program is optional. i.e. our class do not have be part of a package.

Packages are Java’s way of grouping a variety of classes and or interfaces together. The grouping is usually done according to functionality.

Packages act as containers for classes.

5.1.1 The Benefits by Organizing Classes into Packages

- Provides reusability of the class by the other programs existing the same package.

- In packages, classes can be unique compared with classes in other packages. That is, two classes in two different packages can have the same name. They may be referred by their fully qualified name, comprising the package name and the class name.

- Packages provide a way to “hide” classes thus preventing other programs or packages from accessing classes that are meant for internal use only.

- Packages offers a way of separation between “Design” and “Coding” by designing classes and decide their relationships first and implementing the java code needed for the methods next.

5.1.2 Package

Collection of classes grouped together for achieving reusability under a common name. Packages are a way of grouping a variety of classes together. The grouping is done according to their functionality. Simply, Package is a container of classes.

Packages are two types

1. Java API packages

2. User defined packages

Java API provides a large number of classes grouped into different packages according to functionality like lang, util, io, awt, net, applet. Etc. Each package provides different functionality to our java program.
Ex

Using System Packages

There are two ways of accessing the classes stored in a package. The first approach is to use the “fully qualified name” of the class that we want to use. This is achieved by using the “dot operator“ between package and class names.

**Ex**: `java.lang.math`

The lang is a package with in the package java and the hierarchy is represented by separating the levels with dots. We might use a class in a number of places in the program or we may like to use many of the classes contained in a package. We may achieve this easily by the following.

import packagename. Classname

Or

import packagename.*

**Note**: Import statements and must appear at the top of the file, before any class declarations.

### 5.2 Creating and Importing Packages

While declaring a package, first declare the name of the package using the package keyword followed by a package name. This must be the first line in the .java file.

<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>It include classes for primitive types, strings, math functions, threads and exceptions.</td>
</tr>
<tr>
<td>java.util</td>
<td>Language utility classes such as vectors, hash tables, random numbers, date etc.</td>
</tr>
<tr>
<td>java.applet</td>
<td>Classes for creating and implementing applets.</td>
</tr>
<tr>
<td>java.awt</td>
<td>Includes classes for windows, buttons, lists, menus and so on.</td>
</tr>
<tr>
<td>java.io</td>
<td>Input or output support classes.</td>
</tr>
</tbody>
</table>
5.2.1 Declaring a Package

    package <package_name>;  //Package declaration

    Example : package College;

5.2.2 Using one class from other packages

    The following example illustrates how a program will use class from other packages.

    package x;
    public class Y
    {
        public void display()
        {
            System.out.println("Class Y");
        }
    }

    The above example has a package name ‘x’ and containing a single class ‘Y’. This file should be named Y.java stored in the subdirectory’x’. If we compile this java file, the resultant Y.class will be stored in the same subdirectory.

5.2.3 Adding a class to a Package

    • Define the class and make it public.
    • Place the package statement before the class definition.
    • Store this class file in the package directory.
    • Compile the class file. This will create the class file and place it in the package directory.
    • A non public class to a package also follows the same procedure.
    • When we import the package with .* all the classes existed in that package are automatically imported to our program.

Consider the following example

    package x;
    public class Y
In place of statement “package x;” if we place the statement like “package x.*;” all the classes existed in the ‘package x’ are available to the “class Y”

5.2.4 Creating Package with Multiple Public classes

We also create a package with multiple public classes by considering the following statements.

- Finalize the name of the package first.
- Create a subdirectory with this name under the directory where main source files are stored.
- Create classes that are to be placed in the package in separate source files and declare the package statement as first line of source file.

```java
package package_name;
```

- Switch to the subdirectory created earlier and compile each source file. When completed, the package would contain .class files of all the source files.

5.2.5 Importing Packages

Java system package can be accessed either by using fully qualified class name (or) using short cut approach through the import statement. We use the import statement were there are many references to a particular package or the package name is too long.

```java
import <package_name>.*;
import x.*; //make available all the classes existed in that package to the current program.
```

5.2.6 Packages and Searching

The package has to be accessible (via a classpath) in any one of the following ways.

1. The path to the directory must be absolute, in other words, from the root.

(Or)
2. The path to the directory has to be correct relative to the current directory.

5.2.6 Relative and Absolute Paths

A classpath is a collection of one or more paths. Each path in a classpath is either an absolute path or relative path. An absolute path in absolute path in Unix begins with a forward slash (/). The leading slash indicates that this path is starting from the root directory of the system. Because it’s starting from the root, it doesn’t matter what the current directory is – a directory’s absolute path is always the same. A relative path is one that does not start with a slash.

- When a class is put into a package, its fully qualified name must be used.
- An import statement provides an alternate to a class’s fully qualified name.
- In order for a class to be located, its fully qualified name must have a tight relationship with the directory structure in which it resides.
- A classpath can contain both relative and absolute paths
- An absolute path starts with a / or \
- Only the final directory in a given path will be searched.

5.3 Interfaces

The legal implementations for a class are defined in the interface. i.e. any one who knows what the interface look like (not how they are implemented, but how they can be called and what they return) can rest assured that they can invoke those methods on an instance of your implementing class.

Java does not support multiple inheritance i.e. classes in java cannot have more than one super class. Java provides an alternate approach for multiple inheritance known as INTERFACES to support the concept of multiple inheritance. A java class cannot be subclass of more than one super class but it can implement more than one interface.

5.3.1 Defining a Class

Basically, an interface is a kind of class contains fields and methods. The difference between class and an interface is that an interface define only final fields and Abstract Methods. i.e. the Interfaces do not specify any code to implement these methods and data fields contained only constants.
Syntax: interface <interface_name>
{
    Variable declaration;
    Method declaration;
}

In the above syntax “interface” is the key word and the interface_name is any valid java name like class name. Declaring variables for an interface is
Syntax: static final type VariableName =Value;

Note: All the variables are declared as constants.

Method declaration for an interface will contain only a list of methods without any body statement like declaring functions in “C programming “.
Syntax: Return-type <method_Name>(parameter_list);

Ex

Interface student
{
    static final int stdid=303;
    static final String name="venkat";
    void display();
}

Note: The code for the method is not included in the interface and the method declaration simply ends with a semicolon(;) but a class implements this interface must define the code for the method.

5.3.2 Extending Interfaces

Like classes, interfaces can also be extended. i.e., an interface can be a subinterface to another interface. The new subinterface will inherit all the members of the superinterface. For this we will use the keyword extends.

Syntax: Interface x extends y
{
    Body of x;
5.4 Implementing Interfaces

Interfaces are used as “superclasses” whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface.

Syntax

```java
class <class_name> implements <interface_name>
{
    Body of <class_name>;
}
```

Here a class “implements” the interface.

5.4.1 Various Forms Of Interface Implementation

![Diagram of interface implementation]

![Diagram of interface extension]
5.4.2 Interfaces Vs Packages

A package is just a mechanism for grouping objects, it is very similar to grouping items within a folder or directory on a file system. A class is found within a package, but this does not have an impact on the class’ behavior.

An interface, however, is a .java file that is used (implemented) by another class to tell the outside world that it conforms to a certain specification. For example, you might have a “Runnable” interface that has a “run()” method in it, by having a class that is “Runnable” (implements Runnable) anyone using that class knows that it must have a “run()” method defined. This is used when you have several different classes that have the same interface.

Interfaces have more in common with abstract classes than they do with packages. An interface, by definition, cannot have any implemented methods; an abstract class, in contrast, can define some methods and leave some methods to be implemented by a subclass. Also, a class can implement many interfaces, but can only extend one (abstract) class.

Summary

API stands for Application Programming Interface. The Java API contains classes a programmer can use to build applications and applets.

Short Answer Type Questions

1. What is a package? Write the syntax to define a “package”? 
2. What is the difference between User defined package and Java API package?

3. How can we import a package from other package.

4. What is an interface? Write the syntax to define an “interface”?

5. Write any 4 Java API packages.

**Long Answer Type Questions**

1. Write the benefits or advantages of packages and interfaces.

2. How can we add a class to a package.

3. Write about relative and absolute paths.

4. Write the differences between package and interface.

5. Write the procedure to create package with multiple public classes.
6.1 Exception Handling fundamentals.
6.2 Exception types
6.3 The try, catch, throw, throws and finally keywords.
6.4 Java’s built in exceptions.

Learning Objectives

After studying this unit, the student will be able to understand

- What is an Exception
- Types of Exceptions.
- Know about usage of try, catch, throw, throws and finally keywords
- Know about Java built in exceptions.

6.1 Exception Handling fundamentals

A Software engineer may also commit several errors while designing the project or developing the code. These errors are also called bugs and the process of removing bugs is called debugging. In java there are three types of errors.

1. Compile time errors: A compile-time error happens when the program is being compiled. Generally, compile-time errors are syntax errors;
and they are caught by the compiler. An example of a compile-time error might be leaving out a semi-colon or writing the names of predefined methods wrongly etc.,

2. Run time errors: Run-time errors occur at run-time; generally, the program compiles but does not run correctly. Example, insufficient memory to store something or inability of the microprocessor to execute some statement comes under run-time error.

3. Logical errors: The programmer might be using a wrong formula or the design of the program itself is wrong. Logical errors are not detected either by Java compiler or JVM. The programmer is solely responsible for them.

6.1.1 Exceptions

An exception is a problem that arises during the execution of a program. It is a run time error. An exception can occur for many different reasons, including the following.

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications, or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

6.2 Exception types

There are two types of exceptions named as Checked Exceptions and Unchecked Exceptions.

All exceptions occur only at runtime but some exceptions are detected at compile time and some others at runtime. The exceptions that are checked at compilation time by the java compiler are called checked exceptions while the exceptions that are checked by the JVM are called unchecked exceptions.

Unchecked exceptions and errors are considered as unrecoverable and the programmer cannot do anything when they occur. The programmer can write a java program with unchecked exceptions and errors and can compile the program. He can see their effect only when he runs the program. So, Java compiler allows him to write a java program without handling the unchecked exceptions and errors. In case of checked exceptions, the programmer should either handle them or throw them without handling them.
For example consider the following statement, public static void main(String args[]) throws IOException, in this the IOException is a Checked Exception. So, we threw it out of main() method without handling it. This is done by throws clause written after main() method in the above statement.

6.2.1 Exception Hierarchy

![Exception Hierarchy Diagram]

Throwable is a class that represents all errors and exceptions which may occur in Java. Exception is the super class of all exceptions in Java. The difference between error and an exception is, an exception is an error which can be handled. It means when an exception happens, the programmer can do something to avoid any harm. But an error is an error which cannot be handled. It happens and the programmer cannot do anything.

**Common checked exceptions are**
- Class Not Found Exception
- IOException
- EOF Exception
- FileNotFoundException
- NoSuchMethodException

**Common unchecked exceptions**
- ArithmeticException
• Illegal Argument Exception
• Number Format Exception
• Index Out Of Bounds Exception
• Array Index Out Of Bounds Exception
• String Index Out Of Bounds Exception
• Null Pointer Exception

6.3 The try, catch, throw, throws and finally keywords.

When there is an exception, the user data may be corrupted. This should be tackled by the programmer by carefully designing the program. For this, user should perform the following 3 steps.

Step1: The programmer should observe the statements in his program where there may be a possibility of exceptions. Such statements should be written inside a try block. A try block looks like as follows.

```java
try
{
  Statements;
}
```

Step2: The programmer should write the catch block where he should display the exception details to the user. This helps the user to understand that there is some error in the program. The programmer should also display a message regarding what can be done to avoid this error. Catch block looks like as follows.

```java
catch(ExceptionClass ref)
{
  Statements;
}
```

Exception details are available in exception stack. The ref object will refer the exception stack for getting the information regarding an exception. The details of the exception can be print using System.out.println(ref); statement or printStackTrace() method of Throwable class, which fetches exception details from the exception stack and displays them.
Step3: Finally, the programmer should perform clean up operations like closing the files and terminating the threads. The programmer should write this code in the finally block. Finally block looks like as follows:

```java
finally
{
    Statements;
}
```

The specialty of finally block is that the statements inside the finally block are executed irrespective of whether there is an exception or not. This ensures that all the opened files are properly closed and all the running threads are properly terminated. So, the data in the files will not be corrupted and the user is at the safe-side.

6.3.1 Example Programs

A Program Showing How the JVM throws an Exception at runtime

```java
public class DivideException
{
    public static void main(String[] args)
    {
        int a=100,b=4;
        System.out.println(a/b);
        int c=100,d=0;
        System.out.println(c/d);
    }
}
```

Type the above program in notepad editor and save it with a name “DivideException.java”.

After saving the file, first compile the program and then execute the program as follows.
In the first System.out.println you can perform the division between 100 and 4 so it gives the result as 25, in the second System.out.println you can perform the division between 100 and 0 then the JVM throws the exception / by zero because we can not perform the division having 0 as denominator. In the above program the exception is thrown by JVM. By seeing the output the user does not know what is the meaning of that java.lang.ArithmeticException. So, for easy understanding user can handle exceptions thrown by JVM by using exception handling mechanism.

Program shows how the user can handle the exception thrown by JVM

```java
public class DivideException {
    public static void main(String[] args) {
        try {
            int a=100, b=4;
            System.out.println(a/b);
            int c=100, d=0;
            System.out.println(c/d);
        }
    }
}
```
catch(ArithmeticException e)
{
    System.out.println(e);
    System.out.println("don’t give the denominator value as zero for division");
}
}

The output of the above program is as follows.

![Output Image]

**Fig 6.2**

**An example program shows how the finally block will be executed.**

```java
public class ExcepTest
{
    public static void main(String args[])
    {
        int a[] = new int[2];
        try
        {
            System.out.println("Access element three :" + a[3]);
        }
```

```
catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("Exception thrown :" + e);
}

Finally
{
    a[0] = 6;
    System.out.println("First element value: " + a[0]);
    System.out.println("The finally statement is executed");
}

The Output for the above program is as follows.

```
Exception thrown : java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

Fig 6.3

In the program, we declare the array a with two locations but in the try block we want to print the a[3] location element. So, the user handles ArrayIndexOutOfBoundsException that is caught by catch block. After handling the exception the control go to the finally block for performing final operations. In the finally block we print the a[0] location value.

The main purpose of final block is, Code outside the finally block is not guaranteed to execute. Let’s say some sort of unchecked exception happened
that wasn’t caught, or somewhere in the try block you execute the return statement; then code below the finally block would not be executed. Code inside the finally block is always executed, even if an exception was raised and left uncaught, or if the method returned abruptly. Usually ‘finally’ contains connection closing statements, save operations and file input, output close operations.

6.3.2 Throw and throws clause

Before catching an exception it is must to be thrown first. This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception. A single argument is required by the throw statement i.e. a throwable object. As mentioned earlier Throwable objects are instances of any subclass of the Throwable class.

For example, throw new VeryFastException();

For instance the example below shows how to throw an exception. Here we are trying to divide a number by zero so we have thrown an exception here as “throw new MyException("can’t be divided by zero");”

```java
class MyException extends Exception
{
    public MyException(String msg)
    {
        super(msg);
    }
}

public class ThrowEx
{
    static int divide(int first, int second) throws MyException
    {
        if(second==0)
            throw new MyException("can’t be divided by zero");
        return first/second;
    }
}
```
public static void main(String[] args) {
    try {
        System.out.println(divide(4,0));
    }
    catch (MyException e) {
        e.printStackTrace();
    }
}

Output for the above program is as follows.

Whenever we want to force an exception then we use throw keyword. The throw keyword (note the singular form) is used to force an exception. It can also pass a custom message to your exception handling module. Moreover throw keyword can also be used to pass a custom message to the exception handling module i.e. the message which we want to be printed. For instance in the above example we have used -
Throw new MyException ("can’t be divided by zero");

Whereas when we know that a particular exception may be thrown or to pass a possible exception then we use throws keyword. Point to note here is that the Java compiler very well knows about the exceptions thrown by some methods so it insists us to handle them. We can also use throws clause on the surrounding method instead of try and catch exception handler. For instance in the above given program we have used the following clause which will pass the error up to the next level -

```java
Static int divide(int first, int second) throws MyException{
```

### 6.4 Java’s built in exceptions

Java defines several exception classes inside the standard package java.lang. The most general of these exceptions are subclasses of the standard type RuntimeException. Since java.lang is implicitly imported into all Java programs, most exceptions derived from RuntimeException are automatically available.

Java defines several other types of exceptions that relate to its various class libraries. Following is the list of Java Unchecked RuntimeException.

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Exception</td>
<td>Arithmetic error, such as divide-by-zero.</td>
</tr>
<tr>
<td>Array Index Out Of Bounds Exception</td>
<td>Array index is out-of-bounds.</td>
</tr>
<tr>
<td>Array Store Exception</td>
<td>Assignment to an array element of an incompatible type.</td>
</tr>
<tr>
<td>Class Cast Exception</td>
<td>Invalid cast.</td>
</tr>
<tr>
<td>Illegal Argument Exception</td>
<td>Illegal argument used to invoke a method.</td>
</tr>
<tr>
<td>Illegal Monitor State Exception</td>
<td>Illegal monitor operation, such as waiting on an unlocked thread.</td>
</tr>
<tr>
<td>Illegal State Exception</td>
<td>Environment or application is in incorrect state.</td>
</tr>
<tr>
<td>Illegal Thread State Exception</td>
<td>Requested operation not compatible with current thread state.</td>
</tr>
</tbody>
</table>
Summary

- An exception is an event, which occurs during the execution of a program, that interrupts the normal flow of the program. It is an error thrown by a class or method reporting an error in code.

- The ‘Throwable’ class is the superclass of all errors and exceptions in the Java language.

### Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Out Of Bounds Exception</td>
<td>Some type of index is out-of-bounds.</td>
</tr>
<tr>
<td>Negative Array Size Exception</td>
<td>Array created with a negative size.</td>
</tr>
<tr>
<td>Null Pointer Exception</td>
<td>Invalid use of a null reference.</td>
</tr>
<tr>
<td>Number Format Exception</td>
<td>Invalid conversion of a string to a numeric format.</td>
</tr>
<tr>
<td>Security Exception</td>
<td>Attempt to violate security.</td>
</tr>
<tr>
<td>String Index Out Of Bounds</td>
<td>Attempt to index outside the bounds of a string.</td>
</tr>
<tr>
<td>Unsupported Operation Exception</td>
<td>An unsupported operation was encountered.</td>
</tr>
<tr>
<td>Class Not Found Exception</td>
<td>Class not found.</td>
</tr>
<tr>
<td>Clone Not Supported Exception</td>
<td>Attempt to clone an object that does not implement the Cloneable interface.</td>
</tr>
<tr>
<td>Illegal Access Exception</td>
<td>Access to a class is denied.</td>
</tr>
<tr>
<td>Instantiation Exception</td>
<td>Attempt to create an object of an abstract class or interface.</td>
</tr>
<tr>
<td>Interrupted Exception</td>
<td>One thread has been interrupted by another thread.</td>
</tr>
<tr>
<td>No Such Field Exception</td>
<td>A requested field does not exist.</td>
</tr>
<tr>
<td>No Such Method Exception</td>
<td>A requested method does not exist.</td>
</tr>
</tbody>
</table>
• Exceptions are broadly classified as ‘checked exceptions’ and ‘unchecked exceptions’. All RuntimeExceptions and Errors are unchecked exceptions. Rest of the exceptions are called checked exceptions. Checked exceptions should be handled in the code to avoid compile time errors.

• Exceptions can be handled by using ‘try-catch’ block. Try block contains the code which is under observation for exceptions. The catch block contains the remedy for the exception. If any exception occurs in the try block then the control jumps to catch block.

• If a method doesn’t handle the exception, then it is mandatory to specify the exception type in the method signature using ‘throws’ clause.

• We can explicitly throw an exception using ‘throw’ clause.

• An exception is a subclass of the Exception/Error class, both of which are subclasses of the Throwable class. Java exceptions are raised with the throw keyword and handled within a catch block.

• The Throwable class provides a String variable that can be set by the subclasses to provide a detail message that provides more information of the exception occurred. All classes of throwables define a one-parameter constructor that takes a string as the detail message.

• The class Throwable provides getMessage() function to retrieve an exception. It has a printStackTrace() method to print the stack trace to the standard error stream.

• The java code that you think may produce an exception is placed within a try block for a suitable catch block to handle the error. If no exception occurs the execution proceeds with the finally block else it will look for the matching catch block to handle the error. Again if the matching catch handler is not found execution proceeds with the finally block and the default exception handler throws an exception.

• If an exception is generated within the try block, the remaining statements in the try block are not executed.

• Exceptions thrown during execution of the try block can be caught and handled in a catch block. On exit from a catch block, normal execution continues and the finally block is executed.

• A finally block is always executed, regardless of the cause of exit from the try block, or whether any catch block was executed. Generally finally block is used for freeing resources, cleaning up, closing connections etc. If the finally block executes a control transfer statement such as a return or a break statement,
then this control statement determines how the execution will proceed regardless of any return or control statement present in the try or catch.

- A try block can be followed by multiple catch blocks.

- Whenever we want to force an exception then we use throw keyword. The throw keyword (note the singular form) is used to force an exception. It can also pass a custom message to your exception handling module.

- When we know that a particular exception may be thrown or to pass a possible exception then we use throws keyword.

- Java defines several exception classes inside the standard package java.lang. Some of the built in exceptions are like ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException etc.

### Short Answer Type Questions

1. What is debugging?
2. What are compile time error?
3. What are run time errors?
4. What are logical errors?
5. What are three types of errors?
6. What is an exception?
7. What are two exception types?

### Long Answer Type Questions

1. Write about three types of errors.
2. Write about exception types.
3. Explain exception handling in three steps.
4. Explain an example of exception handling in the case of division by zero.
5. Write about some Java’s built in exceptions.
Structure

7.1 Concept of threads
7.2 Multithreading and Multitasking
7.3 The Thread class and its methods
7.4 Writing simple program involving threads.

Learning Objectives

After studying this unit, you will be able to understand

• Threads
• Multithreading and Multitasking
• Difference between Multithreading and Multitasking
• Writing programs by using threads

7.1 Concept of Threads

Those who are familiar with the modern operating systems such as windows XP and windows 7 may recognize that they can execute several programs simultaneously i.e. editing a document in MS-Word and listening songs from a system etc., This ability is known as multitasking. In system’s terminology, it is called multithreading.
Multithreading is a conceptual programming paradigm where a program (process) is divided into two or more programs (processes), which can be implemented at the same time in parallel. For example, one subprogram can display an animation on the screen while another may build the next animation to be displayed. This is something similar to dividing a task into subtasks and assigning them to different people for execution independently and simultaneously.

```
class ABC {
    ...............  
    ...............  
    ...............  
    ...............  
    ...............  
    ...............  
}
```

**Fig. 7.1 Single threaded program**

In most of our computers, we have only a single processor and therefore, in reality, the processor is doing only one thing at a time. However the processor switches between the processes so fast that it appears to human beings that all of them are being done simultaneously.

Java programs that we have seen and discussed so far contain only a single sequential flow of control. This is what happens when we execute a normal program. The program begins, runs through a sequence of executions, and finally ends. At any given point of time, there is only one statement under execution.

A thread is similar to a program that has a single flow of control. It has a beginning, a body, and an end, and executes commands sequentially. In fact, all main programs in our earlier examples can be called single-threaded programs. Every program will have at least one thread as shown in Fig.7.1.
A unique property of Java is its support for multithreading. That is, Java enables us to use multiple flows of control in developing programs. Each flow of control may be thought of as a separate tiny program (or module) known as a thread that runs in parallel to others as shown in Fig. 7.2. A program that contains multiple flows of control is known as a multithreaded program. Fig. 7.2 illustrates a Java program with four threads, one main and three others. The main thread is actually the main method module, which is designed to create and start the other three threads, namely A, B, and C.

Once initiated by the main thread, the threads A, B, and C run concurrently and share the resources jointly. It is like people living in joint families and sharing certain resources among all of them. The ability of a language to support multithreads is referred to as concurrency. Since threads in Java are subprograms of a main application program and share the same memory space, they are known as lightweight threads or lightweight processes.

It is so important to remember that ‘threads running in parallel’ does not really mean that they actually run at the same time. Since all the threads are running on a single processor, the flow of execution is shared between the threads. The Java interpreter handles the switching of control between the threads in such a way that it appears they are running concurrently.
Multithreading is a powerful programming tool that makes Java distinctly different from its fellow programming languages. Multithreading is useful in a number of ways. It enables programmers to do multiple things at one time. They can divide a long program (containing operations that conceptually concurrent) into threads and execute them in parallel. For example, we can send tasks such as printing into the background and continue to perform some other task in the foreground. This approach would considerably improve the speed of our programs.

Threads are extensively used in Java-enabled browsers such as HotJava. These browsers can download a file to the local computer, display a web page in the window, output another web page to a printer and so on.

Any application we are working on that requires two or more things to be done at the same time is probably a best one for use of threads.

### 7.2 Multithreading and Multitasking

<table>
<thead>
<tr>
<th>Multithreading</th>
<th>Multitasking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. It is a programming concept in which a program or a process is divided into two or more subprograms or threads that are executed at the same time in parallel.</td>
<td>1. It is an operating system concept in which multiple tasks are performed simultaneously.</td>
</tr>
<tr>
<td>2. It supports execution of multiple parts of a single program simultaneously.</td>
<td>2. It supports execution of multiple programs simultaneously.</td>
</tr>
<tr>
<td>3. The processor has to switch between different parts or threads of a program.</td>
<td>3. The processor has to switch between different programs are processes.</td>
</tr>
<tr>
<td>4. It is highly efficient.</td>
<td>4. It is less efficient in comparison to multithreading.</td>
</tr>
<tr>
<td>5. A thread is the smallest unit in multithreading.</td>
<td>5. A program or process is the smallest unit in a multitasking environment.</td>
</tr>
<tr>
<td>6. It helps in developing efficient programs.</td>
<td>6. It helps in developing efficient operating systems.</td>
</tr>
<tr>
<td>7. It is cost-effective in case of context switching.</td>
<td>7. It is expensive in case of context switching.</td>
</tr>
</tbody>
</table>
7.3 The threads class and its methods

Creating threads in Java is simple. Threads are implemented in the form of objects that contain a method called `run()`. The `run()` method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the thread’s behavior can be implemented. A typical `run()` would appear as follows.

```java
public void run()
{
    ......................
    ...................... (statement for implementing thread)
    ......................
}
```

The `run()` method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initializing it with the help of another thread method called `start()`.

A new thread can be created in two ways.

1. **By creating a thread class**: Define a class that extends `Thread` class and override its `run()` method with the code required by the thread.

2. **By converting a class to a thread**: Define a class that implements `Runnable` interface. The `Runnable` interface has only one method, `run()`, that is to be defined in the method with the code to be executed by the thread.

The approach to be used depends on what the class we are creating requires. If it requires to extend another class, then we have no choice but to implement the `Runnable` interface, since Java classes cannot have two superclasses.

7.4 Writing simple program involving threads

```java
class A extends Thread{
    public void run(){
        for (int i=1;i<=5;++i)
            System.out.println("form thread A : i="+i);
        System.out.println("exit from A");
    }
}
```
class B extends Thread{
    public void run(){
        for (int i=1;i<=5;++i)
            System.out.println("form thread B :i="+i);
            System.out.println("exit from B");}
}
class C extends Thread{
    public void run(){
        for (int i=1;i<=5;++i)
            System.out.println("form thread C :i="+i);
            System.out.println("exit from C");}
}
class ThreadTest
{public static void main(String args[])
{
    new A().start();
    new B().start();
    new C().start();
}
}Output : 1

Form thread A :i=1
Form thread A :i=2
Form thread A :i=3
Form thread A :i=4
Form thread A :i=5

Exit from A

Form thread B :i=1
Form thread B :i=2
Form thread B :i=3
Form thread B : i=4
Form thread B : i=5

Exit from B
Form thread C : i=1
Form thread C : i=2
Form thread C : i=3
Form thread C : i=4
Form thread C : i=5

Exit from C

Output: 2
Form thread A : i=1
Form thread A : i=2
Form thread A : i=3
Form thread A : i=4
Form thread A : i=5

Exit from A
Form thread B : i=1
Form thread B : i=2
Form thread B : i=3
Form thread B : i=4
Form thread C : i=1
Form thread C : i=2
Form thread C : i=3
Form thread C : i=4
Form thread C : i=5

Exit from C

Exit from B
Summary

A thread is a single line of execution with in a program. Multiple threads can run concurrently in a single program. A Thread is created by either by subclassing the Thread class Or implementing the runnable interface.

Short Answer Type Questions

1. What is multitasking?
2. What is multithreading?
3. What is meant by lightweight threads?

Long Answer Type Questions

1. Write the difference between multithreading and multitasking.
2. Write a simple program by using threads.
After studying this unit, you will be able to understand

- What is an Applet
- Differences between applet and a Java program
- Methods of an applet
- Creating and running a simple applet

8.1 Introduction to Java Applets

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API (Application Program Interface) at its disposal. There are some important differences between an applet and a standalone Java application are
• An applet is a Java class that extends the java.applet.Applet class.

• A main() method is not invoked on an applet, and an applet class will not define main().

• Applets are designed to be embedded within an HTML page.

• When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user’s machine.

• A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.

• The JVM on the user’s machine creates an instance of the applet class and invokes various methods during the applet’s lifetime.

• Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

• Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

8.2 Basic methods of the applet class

Life Cycle of an Applet

Four methods in the Applet class give you the framework on which you build any serious applet.

• **Init** : This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

• **Start** : This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

• **Stop** : This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

• **Destroy** : This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
• **Paint**: Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

### 8.3 Applet initialization and Termination

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet’s execution. The viewer may:

- Request information about the author, version and copyright of the applet.
- Request a description of the parameters the applet recognizes.
- Initialize the applet.
- Destroy the applet.
- Start the applet’s execution.
- Stop the applet’s execution.

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

The “Hello, World” applet is complete as it stands. The only method overridden is the paint method.

### 8.4 The HTML Apple Tag

The following is a simple applet named HelloWorldApplet.java:

```java
import java.applet.*;
import java.awt.*;

public class HelloWorldApplet extends Applet {
    public void paint (Graphics g) {
        g.drawString (“Hello World”, 25, 50);
    }
}
```
These import statements bring the classes into the scope of our applet class

- Java.applet.Applet.
- Java.awt.Graphics.

Without those import statements, the Java compiler would not recognize the classes Applet and Graphics, which the applet class refers to.

**The Applet CLASS**

Every applet is an extension of the java.applet.Applet class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

**These include methods that do the following**

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

### 8.5 Create a Simple Applet

Here we are giving step-wise procedure to create and run an applet

First create a java program **hellojava.java**

```java
import java.awt.*;
import java.applet.*;
public class hellojava extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Welcome to Computer students",10,100);
    }
}
```
When this above program is error free, automatically it creates hellojava.class and then create an html tag hellojava.html as follows. The <applet> tag is the basis for embedding an applet in an HTML file. Below is an example that invokes an applet.

\[
<\text{HTML}>
<\text{HEAD}>
<\text{TITLE}>
\text{WELCOME TO JAVA APPLET}\text{}</TITLE>
</TITLE>
</HEAD>
<BODY>
<CENTER>
<H1>WELCOME TO THE WORLD OF APPLETS</H1>
</CENTER>
<br>
<CENTER>
<APPLET CODE= hellojava.class WIDTH=400 HEIGHT=200>
</APPLET></CENTER>
</BODY>
</HTML>
\]

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.
If we use appletviewer

Appletviewer hellojava.html causes to display as follows.

![Applet Viewer: hellojava.class](image1)

**Fig 8.1**

If we use Internet Explorer or Mozilla, causes to display as follows

![Welcome to the world of applets](image2)

**Summary**

You can refer to HTML Applet Tag to understand more about calling applet from HTML. The code attribute of the `<applet>` tag is required. It specifies the Applet class to run. Width and height are also required to specify the initial size of the panel in which an applet runs. The applet directive must be closed with a `</applet>` tag.
If an applet takes parameters, values may be passed for the parameters by adding `<param>` tags between `<applet>` and `</applet>`. The browser ignores text and other tags between the applet tags.

**Short Answer Type Questions**

1. What is an applet?
2. What are the methods in applet?
3. What are the import statements in applet?

**Long Answer Type Questions**

1. What are the differences between an Applet and stand alone java application?
2. Write about the basic methods of the Applet class.
3. Write the step wise procedure to create and run an applet.
9.1 AWT concepts

9.2 Components and containers

9.3 Commonly used AWT classes

9.4 Example for adding two numbers by using AWT

Learning Objectives

After studying this unit, the student will be able to understand

- AWT, GUI, user interface, AWT components and containers
- AWT classes with simple examples

9.1 AWT concepts

AWT stands for Abstract Window Toolkit. It is a portable GUI library among various operating systems for stand-alone applications.

The Abstract Window Toolkit provides many classes for programmers to use. The Java programming language class library provides a user interface toolkit called the Abstract Windowing Toolkit, or the AWT. The AWT is both powerful and flexible.
What is a user interface

The user interface is that part of a program that interacts with the user of the program. User interfaces take many forms. These forms range in complexity from simple command-line interfaces to the point-and-click graphical user interfaces provided by many modern applications.

At the lowest level, the operating system transmits information from the mouse and keyboard to the program as input, and provides pixels for program output. The AWT was designed so that programmers don’t have worry about the details of tracking the mouse or reading the keyboard, nor attend to the details of writing to the screen. The AWT provides a well-designed object-oriented interface to these low-level services and resources.

Because the Java programming language is platform-independent, the AWT must also be platform-independent. The AWT was designed to provide a common set of tools for graphical user interface design that work on a variety of platforms. The user interface elements provided by the AWT are implemented using each platform’s native GUI toolkit, thereby preserving the look and feel of each platform. This is one of the AWT’s strongest points. The disadvantage of such an approach is the fact that a graphical user interface designed on one platform may look different when displayed on another platform.

9.2 Components and containers

A graphical user interface is built of graphical elements called components. Typical components include such items as buttons, scrollbars, and text fields. Components allow the user to interact with the program and provide the user with visual feedback about the state of the program. In the AWT, all user interface components are instances of class Component or one of its subtypes.

Components do not stand alone, but rather are found within containers. Containers contain and control the layout of components. Containers are themselves components, and can thus be placed inside other containers. In the AWT, all containers are instances of class Container or one of its subtypes.

Especially, components must fit completely within the container that contains them. This nesting of components (including containers) into containers creates a tree of elements, starting with the container at the root of the tree and expanding out to the leaves, which are components such as buttons.

The illustration in Figure 1 depicts a simple graphical user interface as it would look when displayed under Windows 7.
9.2.1 Types of Components

The AWT provides nine basic non-container component classes from which a user interface may be constructed. (Of course, new component classes may be derived from any of these or from class Component itself.) These nine classes are class Button, Canvas, Checkbox, Choice, Label, List, Scrollbar, TextArea, and TextField. Figure 2 depicts an instance of each class.
9.2.2 Types of containers

The AWT provides four container classes. They are class Window and its two subtypes — class Frame and class Dialog — as well as the Panel class. In addition to the containers provided by the AWT, the Applet class is a container — it is a subtype of the Panel class and can therefore hold components. Brief descriptions of each container class provided by the AWT are provided below.

Window

A top-level display surface (a window). An instance of the Window class is not attached to nor embedded within another container. An instance of the Window class has no border and no title.

Frame

A top-level display surface (a window) with a border and title. An instance of the Frame class may have a menu bar. It is otherwise very much like an instance of the Window class.

Dialogue

A top-level display surface (a window) with a border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.

Panel

A generic container for holding components. An instance of the Panel class provides a container to which to add components.

Hierarchical components of AWT

After having the basic idea of Java AWT, let us know how many component classes exist with java.awt package. This knowledge is essential to go further in this unit.

As you can observe from the above hierarchy, Component is the super class of all Java components and is declared as abstract. That is, we cannot create objects of Component class directly.
9.3 Commonly used AWT Classes

The Abstract Window Toolkit (AWT) contains numerous classes and methods that allow you to create and manage windows. The AWT classes are contained in the ‘java.awt’ package. Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use.

9.3.1 Creating a frames

Before adding the components that make up a user interface, the programmer must create a container. When building an application, the programmer must first create an instance of class Window or class Frame. When building an applet, a frame (the browser window) already exists. Since the Applet class is a subtype of the Panel class, the programmer can add the components to the instance of the Applet class itself.

The code in Listing 1 creates an empty frame. The title of the frame (“Example 1”) is set in the call to the constructor. A frame is initially invisible and must be made visible by invoking its show() method.

```java
import java.awt.*;

public class Example1 {

```
public static void main(String Args[]) {
    Frame f = new Frame("Example 1");
    f.show();
}

When we compile the above Example1.java, the empty frame will be displayed as follows.

```
D:\java>javac Example1.java
Note: Example1.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\java>java Example1
```

![Example 1 Output]

9.3.2 Adding components to a container

To be useful, a user interface must consist of more than just a container it must contain components. Components are added to containers via a container’s add() method. There are three basic forms of the add() method. The method to use depends on the container’s layout manager.

The code is for the creation of two buttons. The creation is performed in the init() method because it is automatically called during applet initialization. Therefore, no matter how the program is started, the buttons are created, because init() is called by either the browser or by the main() method. Figure below contains the resulting applet.
import java.awt.*;

public class Example3 extends java.applet.Applet
{
    public void init()
    {
        add(new Button("One"));
        add(new Button("Two"));
    }

    public Dimension preferredSize()
    {
        return new Dimension(200, 100);
    }

    public static void main(String [] args)
    {
        Frame f = new Frame("Example 3");
        Example3 ex = new Example3();
        ex.init();
        f.add("Center", ex);
        f.pack();
        f.show();
    }
}

When we run the above code, the output will be as follows:

![Example 3 Window]

9.4 Example for adding two numbers by using AWT

The following ex1.java explains adding two numbers by using AWT:

import java.awt.*;
import java.awt.event.*;
class ex1 extends Frame implements ActionListener
{  
Label l1,l2,l3,l4;
TextField t1,t2;
Button b1,b2;
ex1()
  
{  
super(“Addition of two numbers”);
setLayout(null);
setSize(500,400);
setVisible(true);
l1=new Label(“Enter the first no:”);
l2=new Label(“Enter the second no:”);
l3=new Label(“The sum is:”);
l4=new Label(null);
t1=new TextField();
t2=new TextField();
b1=new Button(“ADD”);
b2=new Button(“Clear”);
l1.setBounds(100,50,120,20);
add(l1);
t1.setBounds(240,50,50,20);
add(t1);
l2.setBounds(100,80,130,20);
add(l2);
t2.setBounds(240,80,50,20);
add(t2);
l3.setBounds(100,110,100,20);
add(l3);
add(l3);
l4.setBounds(210,110,60,20);
add(l4);
b1.setBounds(200,150,50,20);
add(b1);
b2.setBounds(270,150,50,20);
add(b2);
b1.addActionListener(this);
b2.addActionListener(this);
addWindowListener( new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
        System.exit(0);
    };
});

public void actionPerformed(ActionEvent ae) {
    float a,b,c;
    if(ae.getSource()==b1) {
        a=Float.parseFloat(t1.getText().trim());
b=Float.parseFloat(t2.getText().trim());
c=a+b;
l4.setText(Float.toString(c));
else
{
    t1.setText(null);
    t2.setText(null);
    l4.setText(null);
}

public static void main(String s[])
{
    ex1 ob=new ex1();
}

The output of the above example is

![Image of an addition calculator]

**Summary**

Of course, newcomers should expect some difficulty. Effective graphical user interfaces are inherently challenging to design and implement, and the sometimes complicated interactions between classes in the AWT only make this
task more complex. However, with proper guidance, the creation of a graphical user interface using the AWT is not only possible, but relatively straightforward.

**Short Answer Type Questions**

1. What is AWT?
2. What is an user interface?
3. What are different types of components in AWT?
4. What are different types of containers in AWT?
5. What is window container in AWT?
6. What is Frame container in AWT?
7. What is Dialogue container in AWT?
8. What is panel container in AWT?

**Long Answer Type Questions**

1. Write about nine types of components classes.
2. Write about types of containers.
3. Write about hierarchical order of AWT components.
4. Write a program for creating a frame by using AWT.
5. Write a program for adding a component to a frame by using AWT.
6. Write a program for adding two numbers by using AWT.
10.1 Concepts of Events

Changing the state of an object is known as an event. For example, clicking on button, dragging mouse etc.

In AWT components, we came to know every component (except Panel and Label) generates events when interacted by the user like clicking over a button or pressing enter key etc. Listeners handle the events. Let us know the style (or design pattern) Java follows to handle the events.
The event handling involves four types of classes.

1. Event Sources

   Event sources are components, subclasses of AWT components, capable to generate events. The event source can be a button, TextField or a Frame etc.

2. Event classes

   Almost every event source generates an event and is named by some Java class. For example, the button generates Action Event and Checkbox generates Item Event. All events listed in java.awt.event package.

3. Event Listeners

   The events generated by the GUI components are handled by a special group of classes known as “listeners”. Listener is an interface. Every component has its own listener, say, AdjustmentListener handles the events of scrollbar. Some listeners handle the events of a few components. For example, ActionListener handles the events of Button, TextField, List and Menus. Listeners are from java.awt.event package.

4. Event Adapters

   When a listener includes many abstract methods to override, the coding becomes heavy to the programmer. For example, to close the frame, you override seven abstract methods of WindowListener, in which, infact you are using only one method. To avoid this heavy coding, the designers come with another group of classes known as “adapters”. Adapters are abstract classes defined in java.awt.event package. Every listener that has more than one abstract method has got a corresponding adapter class.
10.2 Event classes and listener interfaces

<table>
<thead>
<tr>
<th>Event Classes</th>
<th>Listener Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action event</td>
<td>ActionListener</td>
</tr>
<tr>
<td>Mouse Event</td>
<td>Mouse listener and Mouse Motion Listener</td>
</tr>
<tr>
<td>Mouse Wheel event</td>
<td>Mouse Wheel Listener</td>
</tr>
<tr>
<td>Key Event</td>
<td>Key Listener</td>
</tr>
<tr>
<td>Item Event</td>
<td>Item Listener</td>
</tr>
<tr>
<td>Text Event</td>
<td>Text Listener</td>
</tr>
<tr>
<td>Adjustment Event</td>
<td>Adjustment Listener</td>
</tr>
<tr>
<td>Window Event</td>
<td>Window Listener</td>
</tr>
<tr>
<td>Component Event</td>
<td>Component Listener</td>
</tr>
<tr>
<td>Container Event</td>
<td>Container Listener</td>
</tr>
<tr>
<td>Focus Event</td>
<td>Focus Listener</td>
</tr>
</tbody>
</table>

Public void set Bounds

(int x-axis, int y- **Steps to perform EventHandling**

Following steps are required to perform event handling

1. Implement the Listener interface and overrides its methods

2. Register the component with the listener

10.3 Commonly used Methods

For registering the component with the listener, many classes provide the registration methods.

For example

- **Button**: Public void addActionListener(ActionListener a) {}
- **MenuItem**: Public void addActionListener(ActionListener a) {}
- **TextField**: Public void addActionListener(ActionListener a) {}
  Public void addTextListener(TextListener a) { }
10.4 Examples

Example-1 simple example of event handling i.e., when we press a mouse button, it will display “welcome”.

```java
import java.awt.*;
import java.awt.event.*;

class AEvent extends Frame implements ActionListener{
    TextField tf;
    AEvent(){
        tf = new TextField();
        tf.setBounds(60,50,170,20);
        Button b = new Button(“click me”);
        b.setBounds(100,120,80,30);
        b.addActionListener(this);
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e){
        tf.setText(“welcome”);
    }

    public static void main(String args[]){
        new AEvent();
    }
}
```
} } 

//end

axis , int width , int height); have been used in the above example

that sets the position of the component it may be button,textfield etc.

The output of the above program is

![Image of program output]

When we make mouse press on “click me”, the output will be as follows

![Image of program output]

Example-2

This is an example of simple ActionListener event handler. Action

listeners are probably the easiest - and most common - event handlers to

implement. You implement an action listener to define what should be done

when an user performs certain operation.
This is a simple program which displays how many number of times a button is clicked by the user. First, here is the code that sets up the TextField, button and numClicks variable.

Public class AL extends Frame implements Window Listener, ActionListener {

    TextField text = new TextField(20);
    Button b;
    private int numClicks = 0;

    In the above example, the event handler class is AL which implements ActionListener.

    We would like to handle the button-click event, so we add an action listener to the button b as below:

    b = new Button("Click me");
    b.addActionListener(this);

    In the above code, Button b is a component upon which an instance of event handler class AL is registered.

    Now, we want to display the text as to how many number of times a user clicked button. We can do this by writing the code as below:

    public void actionPerformed(ActionEvent e) {
        numClicks++;
        text.setText("Button Clicked " + numClicks + " times");

    Now, when the user clicks the Button b, the button fires an action event which invokes the action listener’s actionPerformed method. Each time the user presses the button, numClicks variable is appended and the message is displayed in the text field.

    The AL.java explains simple event handling for number of times button pressed through mouse.

    import java.awt.*;
    import java.awt.event.*;

    public class AL extends Frame implements WindowListener, ActionListener {

TextField text = new TextField(20);
Button b;
private int numClicks = 0;
public static void main(String[] args) {
    AL myWindow = new AL("My first window");
    myWindow.setSize(350, 100);
    myWindow.setVisible(true);
}
public AL(String title) {
    super(title);
    setLayout(new FlowLayout());
    addWindowListener(this);
    b = new Button("Click me");
    add(b);
    add(text);
    b.addActionListener(this);
}
public void actionPerformed(ActionEvent e) {
    numClicks++;
    text.setText("Button Clicked " + numClicks + " times");
}
public void windowClosing(WindowEvent e) {
    dispose();
    System.exit(0);
}
public void windowOpened(WindowEvent e) {}
public void windowActivated(WindowEvent e) { }
public void windowIconified(WindowEvent e) {}  
public void windowDeiconified(WindowEvent e) {}  
public void windowDeactivated(WindowEvent e) {}  
public void windowClosed(WindowEvent e) {}

//end

The output of the program is as follows

After pressing mouse 3 times

Example-3

Below shows how to handle one type of event a program may receive. The new class overrides the action() method provided by the Component class. The action() method responds to the action events that are generated, for example, by the selection of an item from a pop-up list. The action() method requires that two parameters be supplied, an Event instance and an Object instance. The Event instance contains information about the event, including the target of the event (the component that first received the event), the x and y coordinates of the event, and the time when the event occurred. The Object instance holds an event-specific piece of data. For Button objects it contains the text in the button label.
import java.awt.*;

public class Example5 extends java.applet.Applet
{
    TextArea ta = null;

    public void init()
    {
        Panel p;
        setLayout(new BorderLayout());
        p = new Panel();
        ta = new TextArea();
        p.add(ta);
        add("Center", p);
        p = new Panel();
        p.add(new Button("One"));
        p.add(new Button("Two"));
        Choice c = new Choice();
        c.addItem("one");
        c.addItem("two");
        c.addItem("three");
        p.add(c);
        add("South", p);
    }

    public boolean action(Event e, Object o)
    {
        String str = (String)o;
        ta.appendText(str + "\n");
        return false;
    }
}
```java
public static void main(String[] args)
{
    Frame f = new Frame("Example 5");
    Example5 ex = new Example5();
    ex.init();
    f.add("Center", ex);
    f.pack();
    f.show();
}

//end

The output of the program

![Image of a Java GUI window]

when we press the buttons many times as we require, the output may be as follows.
Summary

Some simple examples were given in this unit to understand the concept of event Handling. The examples above do nothing more than display and insert user interface. But there are many Event classes, so that we can use many ways in GUI applications as we require. Nowadays so many websites are also available to give examples. It is, of course, very important that a user interface take action as a result of user input.

Short Answer Type Questions

1. What is an event?
2. What are steps to perform event handling?

Long Answer Type Questions

1. Write about four types of classes involved in event handling.
2. List the event classes and Listener Interfaces.
3. What are commonly used methods in Event handling?
4. Write a program to display a text message “welcome” when we press mouse click.