Core Java

with CC-214 Practicals

BCA Semester-4 | Subject CC-210

- Java Introduction
- Array, Inheritance and Interface
- Package, String and Exception Handling
- Multithreading and Applet

COMPUTER WORLD PUBLICATION
A Division of Live Education System Pvt. Ltd.

BCA | Semester-4

Core Java

with CC-214 Practicals

Subject Code : CC-210 & CC-214 (Practicals)

Dr. Shivang Patel
PhD [Computer Science]
Chimanbhai Patel Institute of Computer Applications,
SG Highway, Ahmedabad.

Prof. Janhavee Mistry
B.Sc (Chemistry), MCA, PhD (Pursuing)
R.B.Institute of Management and
Computer Studies College,
Thakkarbapanagar, Ahmedabad.

Dr. Darshna Patel
PhD [Computer Science]
Navgujarat College of Computer Applications,
Ashram Road, Ahmedabad.

Prof. Vidhi Bhatt
MCA [Master in Computer Application]
Lokmanya College of Computer Applications,
Satellite, Ahmedabad.

COMPUTER® WORLD
PUBLICATION
A Division of Live Education System Pvt. Ltd.

COMPUTER WORLD
43, 5th Floor, SANIDHYA Complex, Nr. M. J. Library,
Opp. Sanyas Ashram, Ashram Road, Ahmedabad-09.
Ph. : +91- 79 26580723, 26580823 | Mobile : 9724011150, 9725022917
URL : www.computerworld.ind.in | e-Mail :info@computerworld.ind.in

Publish By :

COMPUTER World

A Division of Live Education System Pvt. Ltd.

An ISO 9001 : 2008 Certified Company

Kalpesh Patel - kalpesh@computerworld.ind.in

Prepare by : Computer World Research Department

Dr. Shivang Patel, Prof. Janhavee Mistry

Dr. Darshna Patel, Prof. Vidhi Bhatt, Ravindra Parmar

Price : ₹ 185/-

ISBN : 978-93-88092-24-1

Book Code : CEBCA116

Edition : 1st

INDEX

Unit -1

Java Introduction

- Java Introduction
- Java Programming constructs
- Classes and Objects

Unit-1 Java Introduction

1.1 Java Introduction:

1.1.1 Principles of Object oriented language:

In this Java oops concepts, we will learn four major object oriented principles – abstraction, encapsulation, inheritance, and polymorphism. They are also known as basic structure of the object oriented programming language.

Abstraction:

Abstraction is one of the key concepts of object-oriented programming (OOP) languages. Its main goal is to handle difficulties by hiding unnecessary details from the user. Abstraction is selecting data from a larger pool to show only the relevant details to the object. It helps to reduce programming complexity and effort.

In Java, abstraction is accomplished using Abstract classes and interfaces. It is one of the most important concepts of oops.

Ex:- A shape is viewed as a shape rather than its individual components.

Encapsulation:

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Other way to think about encapsulation is, it is a protective class that prevents the data from being accessed by the code outside this class.

Practically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.

As in encapsulation, the data (variable) in a class is hidden from other classes, so it is also known as data-hiding.

Ex:- let's take the example of a medical capsule, where the drug is always safe inside the capsule. Similarly, through encapsulation, the methods and variables of a class are well hidden and safe.

Inheritance:

Inheritance is a mechanism in which one class get the property of another class. For example, a child inherits the character of his/her parents. With inheritance, we can reuse the fields and methods of the existing class which are not private. Hence, inheritance facilitates Reusability and is an important concept of oops. Inheritance in Java can be best understood in terms of Parent and Child relationship, also known

as Super class(Parent) and Sub class(child) OR base class(Parent) and derive class(child) in Java language. So, it's also called as IS-A relationship.

Ex:- circle is sub class get all the relevant property from super class shape.

Polymorphism:

The word polymorphism is used in various contexts and describes situations in which something occurs in several different forms. Polymorphism is a oops concept where one name can have many forms. In computer science, it describes the concept that objects of different types can be accessed through the same interface. Each type can provide its own, independent implementation of this interface.

For example, you have a smartphone for communication. The communication mode you choose could be anything. It can be a call, a text message, a picture message, mail, etc. So, the goal is common that is communication, but their approach is different. This is called Polymorphism.

1.1.2 Java Essentials:

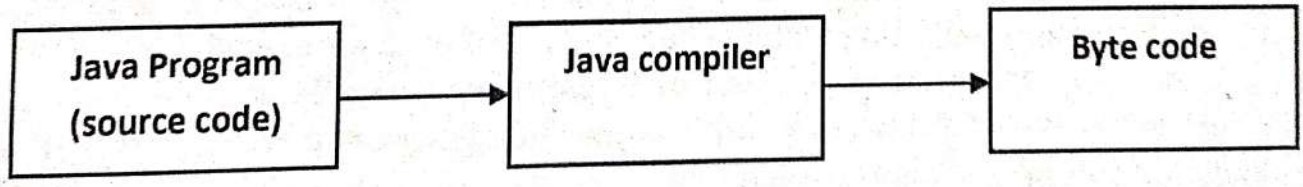
If you are new to programming in the Java language, have some experience with other languages used oops features, and are familiar with things like displaying text or graphics or performing simple calculations. It walks through how to use the Java Platform software to create and run program.

You will learn how to write the applications, how to build a basic user interface that handles simple end user input, how to read data from and write data to files and databases, from this book you have to learn more common programming features available in the Java platform.

1.1.3 Java Virtual Machine:

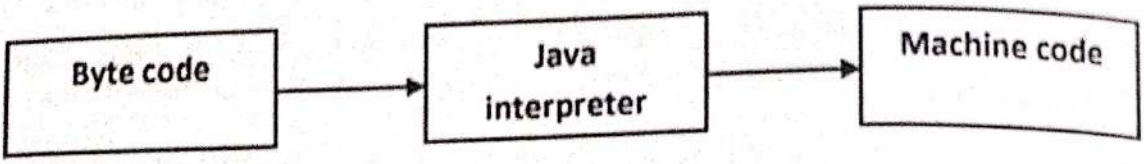
As we know that all programming language compilers convert the source code to machine code.In java, Java compiler convert the source code into Intermediate code is called as bytecode.This machine is called the Java Virtual machine and it exits only inside the computer memory.

The process of compilation



The byte code is not machine specific. The machine code is generated By Java interpreter by acting as an intermediary between the two different machines shown below.

The process of running program



1.1.4 Java Features:

The Java is an object oriented programming language developed by Sun Microsystems of USA in 1991.Program developed in Java can be executed anywhere and on any system.

Features of java listed below:

1. Platform Independent:

Java is Platform independent and supports the feature portability. Java programs can be easily moved from one computer system to another and any where. Changes and upgrades in operating systems, processors and system resources will not force any alteration in Java programs.

2. Object Oriented:

Java is truly object-oriented language.Objects have two sections. The first is Data (instance variables) and the second is methods. Data represents what an object is. A method represents what an object does. In Java, almost everything is an Object. The object model in Java is trouble-free,easy to enlarge, reuse of programming segments and robust.

3. Robust and secure:

Java is a most strong language which provides many securities to make certain reliable code. Many times pointers are the main cause of runtime errors because of improper use of memory. Java eliminates pointer manipulation completely from the language, and therefore eliminates a large source of runtime errors. Java programmers need not remember to de-allocate memory in programs since there is a garbage collection mechanism which handles de-allocation of memory. It provides powerful a robust exception handling mechanism to deal with both expected and unexpected errors at run time.Java also includes the concept of exception handling, which control serious errors and reduces all kind of threat of crashing the system. Security is an important feature of Java and this is the strong reason that programmer use this language for programming on Internet.

4. Used Multithreading:

The execution of two or more sections of a program at the same time and this technique is known as multithreading.Multithreaded means managing multiple tasks simultaneously. Java maintains multithreaded programs. That means we need not wait

for the application to complete one task before starting next task. This feature is helpful for graphic applications.

5. Dynamic and Extensible:

The Java compiler is smart and dynamic. If you are compiling a file that depends on other non-compiled files, then the compiler will try to find and compile them also. The compiler can handle methods that are used before they're declared. It can also determine whether a source code has been changed since the last time it was compiled.

6. High Performance:

Java is a fast-interpreted language.Java has designed for the run-time system can optimize their performance by compiling bytecode to native machine code on the fly (execute immediately after compilation). This is called "just in time" (JIT) compilation.

7. Easy to Learn:

Java is very simple language. Java does not use pointer and header files, go to statements, etc. It eliminates operator overloading, multiple inheritance and pointers.In Java is automatic memory allocation and de-allocation.

1.1.5 Program Structure:

Java environment includes a number of development tools, classes and methods.
The development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

Java Development kit (JDK) – The JDK comes with a set of tools that are used for developing and running Java program.

For compiling and running the program we have to use following commands:

A) javac (Java compiler)

In java, we can use any text editor for writing program and then save that program with —.java extension.

Java compiler convert the source code or program in bytecode and interpreter convert —.java file in —.class file.

Syntax: C:\javac filename.java

If my filename is —abc.javal then the syntax will be

Eg. C:\javac abc.java

B) java(Java Interpreter)

Java interpreter convert bytecode into machine code and —.java file in —.class file.

Syntax: C:\java filename

If my filename is abc.java then the syntax will be
Eg. C:\java abc

A Simple Java Program:

Java offers two flavors of programming, Java applets and Java application.

Java application Program

Let us write your first program with the file name "Application". This program, after successful compilation and run, prints the words "This is My First Java Program" on your display.

1. First of all using any text editor, type Java source file of your program as given below.
2. Now compile the source file-using compiler named Javac that takes your source file and translates its statements into a bytecode file.
3. Run the program using interpreter named Java that takes your bytecode file and translates them into machine code that your computer can understand.

```
/* This is my First Java Application Save the file as Application.Java: same as class name */
Class Application
{
        Public static void main (String args[ ] )
        {
                System.out.println("This is My First Java Application");
        }                                   // main ends here
} // Code ends here
```

Let us see the above program line by line.

Line 1. /* This is my First Java Program. Save this file as Application.Java same as class name */

Comments in Java are similar to C++. For single line comment we use // and for multiline write everything between /* and */ which is ignored by the compiler but Comments allow you to describe the details of the program. This is useful for developing the understandability in your program.

Line 2. Class Application

Second line in the program defines a class named Application using a keyword class.

Line 4. Public static void main (String args[ ])

This is the point from where the program will start the execution. This program starts the execution by calling main () method. In this line public, static, and void all are keywords. May be you are thinking of the meaning of 'keyword'.
Keywords are nothing but some reserved words.

- The public keyword is used to control the access of various class members. If member is public it can be accessed outside the class.
- So we have to declare main () as public because it has to be invoked by the code outside the class when program is executed.
- Static key word allows the main () method to be executed without creating an object of that class,
- And void means main () method does not return any value.
- Parentheses ( )
  It encloses arguments in method definitions or calling and used to define test expressions control statements.
- Braces { }
  It defines blocks of code
- Square bracket [ ]
  It declares array types.
- Semicolons;
  Are used to terminate statements.
- Single dot "."
  It Selects a method from an object and separates package names from subpackags And class names.

Line 6. System.out.println("This is My First Java Application.");

This line prints the string "This is My First Java Application".

- Println ()
  This function is used to display this line. The println () function accepts any string and Display its value on console.

Compiling the Program:

To compile the Example program, execute the compiler, javac, specifying the name of the source file on the command line, as shown here:

C:\>javac Application.java

The javac compiler creates a file called Application.class that contains the bytecode version of the program. The Java bytecode is the intermediate of program.the Java interpreter will execute.

Thus, the output of javac is not code that can be directly executed. To actually run the program, you must use the Java interpreter, called java. To do so, pass the class name Example as a command-line argument, as shown here:

C:\>java Application

When the program is run, the following output is displayed:

This is My First Java Application.

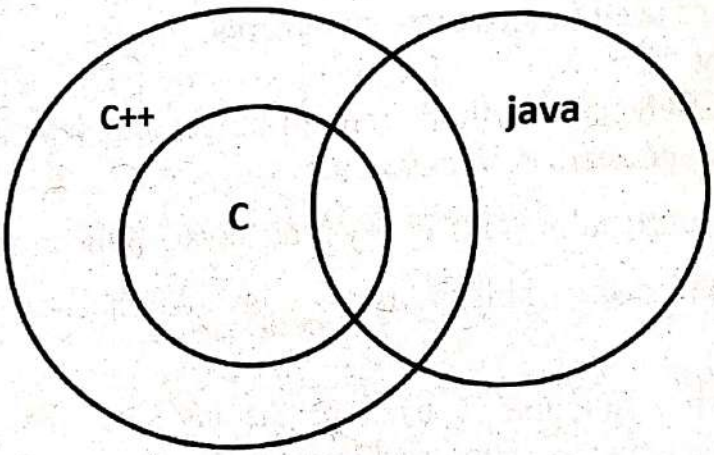
1.1.6 Java Improvements:

Even with the occurrence of other programming languages, Java has continued its widespread popularity since it was introduced 23 years ago. One of the forceful reasons why it's so popular is its platform-independent nature. This means that a Java program can work on any operating system or computer.

Many of the largest web development applications and systems around the world are developed using this language. Amazon, Google, Android, Experian, Twitter, eBay, and Netflix are just a few of the popular companies that use Java.

The latest version of Java Platform Standard Edition (Java SE) is JDK 12 in the Java Community Process.

1.1.7 Difference between Java and C++:



Difference between java and C

- It doesn't include "sizeof" and "typedef".
- It doesn't contain "struct" and "union".
- It doesn't define type modifier keyword auto,extern,register,signed and unsigned.
- It doesn't support pointer type.
- It doesn't have preprocessor #define , #include statement.
- It requires that the functions with no argument must be declared with empty parenthesis and not with void keyword.
- Java adds new operators such as instanceof and >>>
- Java adds new labelled such as break and continue
- Java adds many features required for object oriented programming.

Difference between java and C++:

- It doesn't support operator overloading
- It doesn't have template class in c++
- It doesn't support multiple inheritance of classes.this is accomplished using using "interface"
- It doesn't support global variable.every variable and methods is declared within a class.
- It doesn't use pointer.
- It has replaced the destructor function with a finalize() method.
- There are no header files in java.

Comparison in Java and C++:

Java	C++
Java is true Object-oriented language.	C++ is basically C with Object-oriented extension.
Java does not support operator overloading.	C++ supports operator overloading.
It supports labels with loops and statement blocks	It supports goto statement.
Java does not have template classes as in C++.	C++ has template classes.
Java compiled into byte code for the Java Virtual Machine. The source code is independent on operating system.	Source code can be written to be platform independent and written to take advantage of platform.C++ typically compiled into machine code.
Java does not support multiple inheritance of classes but it supports interface.	C++ supports multiple inheritance of classes.
Runs in a protected virtual machine.	Exposes low-level system facilities.
Java does not support global variable. Every variable should declare in class.	C++ support global variable.
Java does not use pointer.	C++ uses pointer.
It Strictly enforces an object oriented programming paradigm.	It Allows both procedural programming and object-oriented programming.
There are no header files in Java.	We have to use header file in C++.

1.1.8 Installation of JDK 1.7:

Java is a computer programming language that is concurrent, class-based and object-oriented. It was originally developed by James Gosling at Sun Microsystems. Java applications are compiled to bytecode (class file) that can run on any Java virtual machine (JVM).

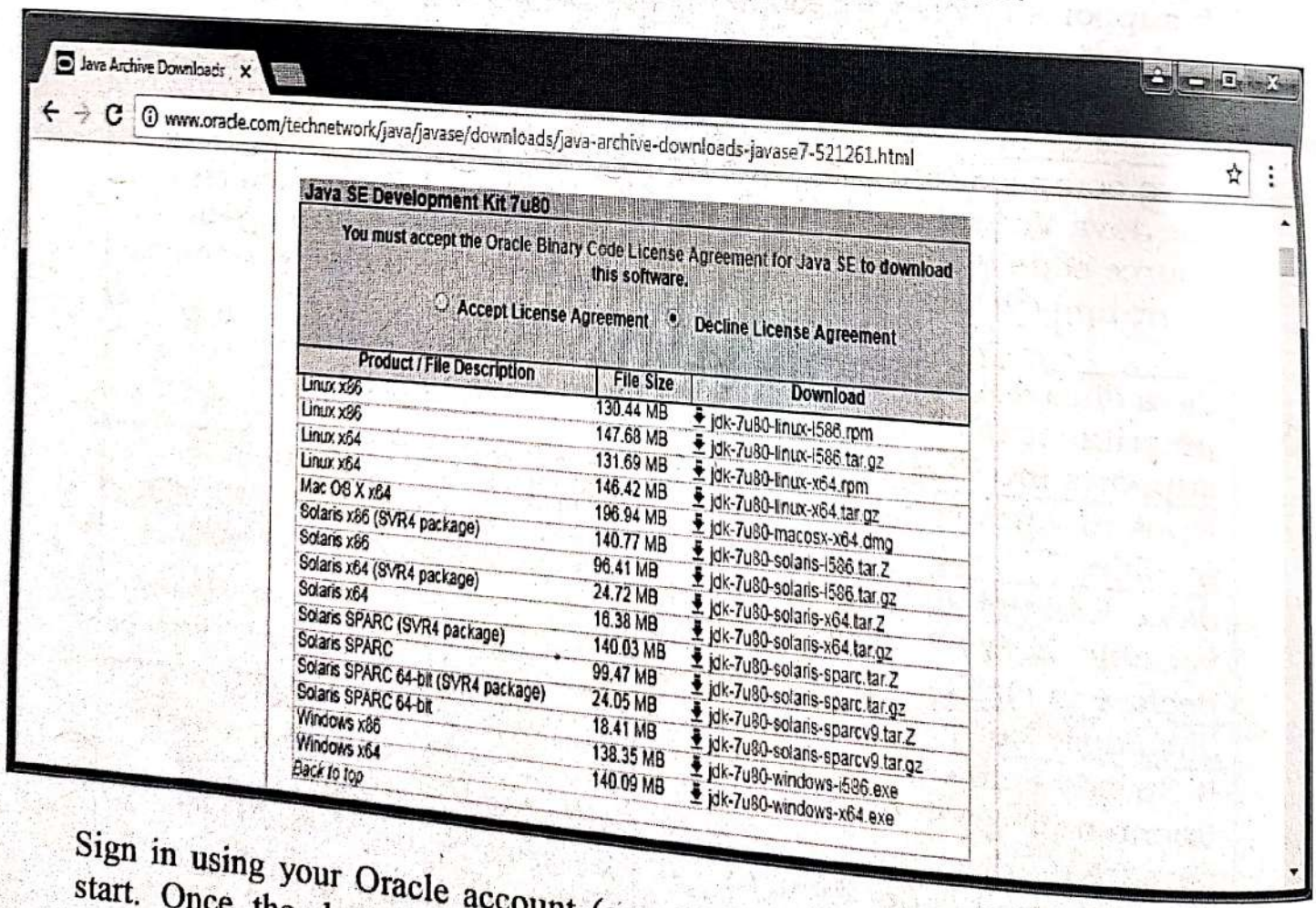
Following steps will show you how to setup and configure Java 1.7 on Windows so you can develop and run Java code.

JDK Download & Install:

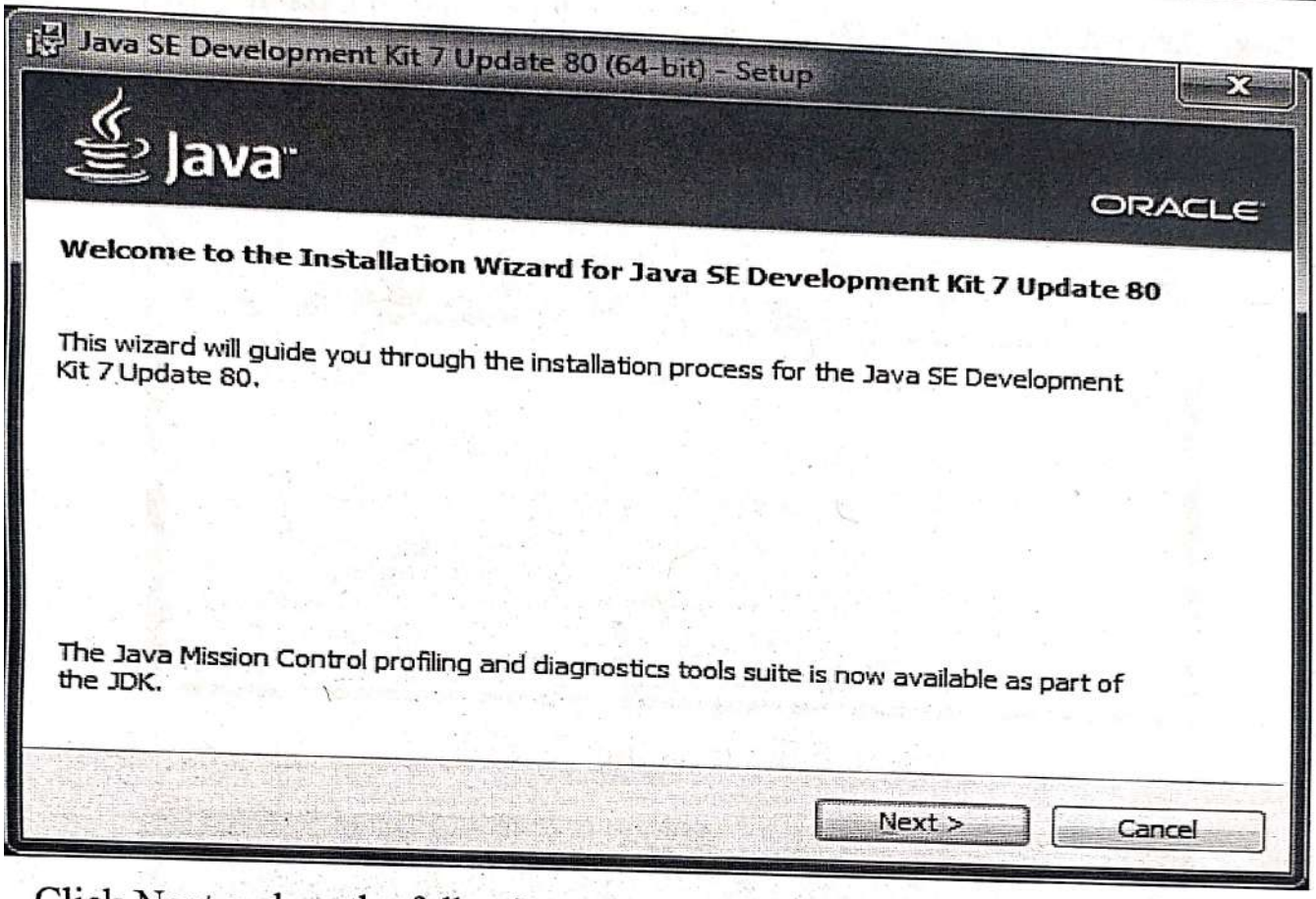
Java can be obtained from the Oracle Java download page. There are a number of different Java packages available, Here,we will be installing Java Standard Edition (SE) on Windows.

In order to be able to compile Java code, we need the Java Development Kit (JDK) package that comes with a Java compiler. The JDK package also comes with a Java runtime environment (JRE) that is needed to run compiled Java code.

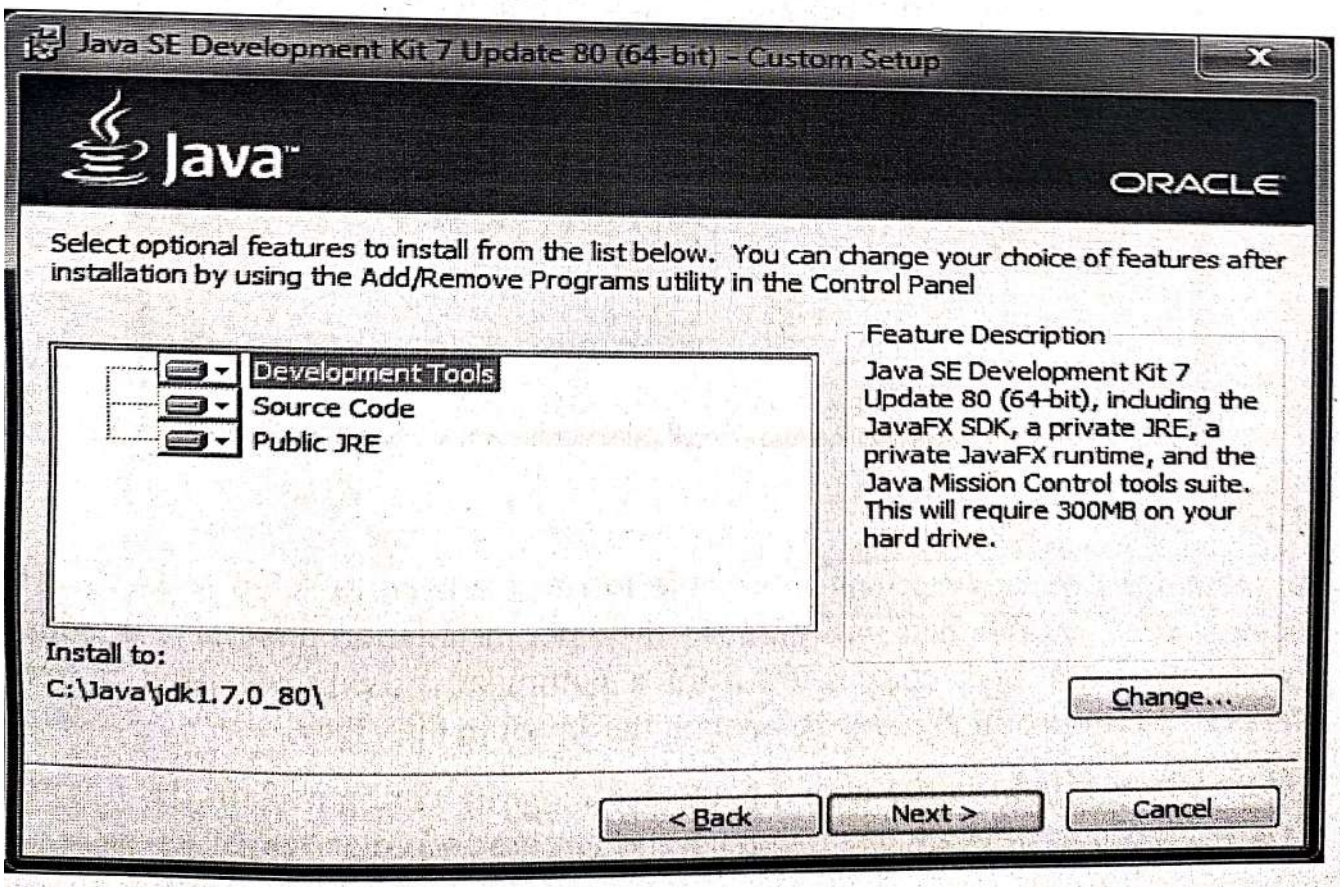
Accept the License Agreement and pick the correct download for your operating system. In this example, we will use the Windows 64 bit version.



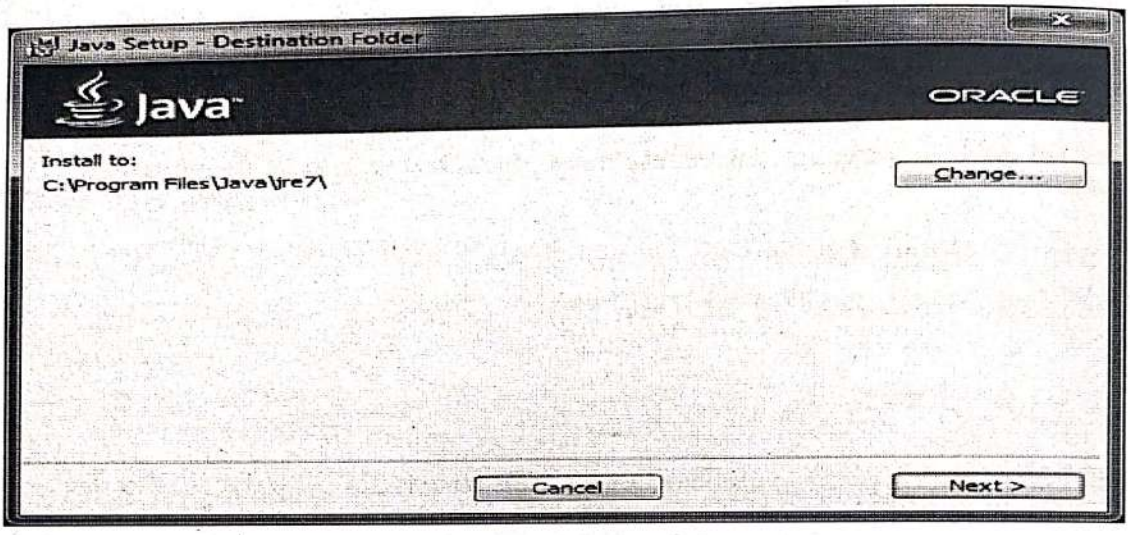
Sign in using your Oracle account (or create a new one) and the download should start. Once the download is complete, locate the jdk-7u80-windows-x64 file and double-click to run the installer.



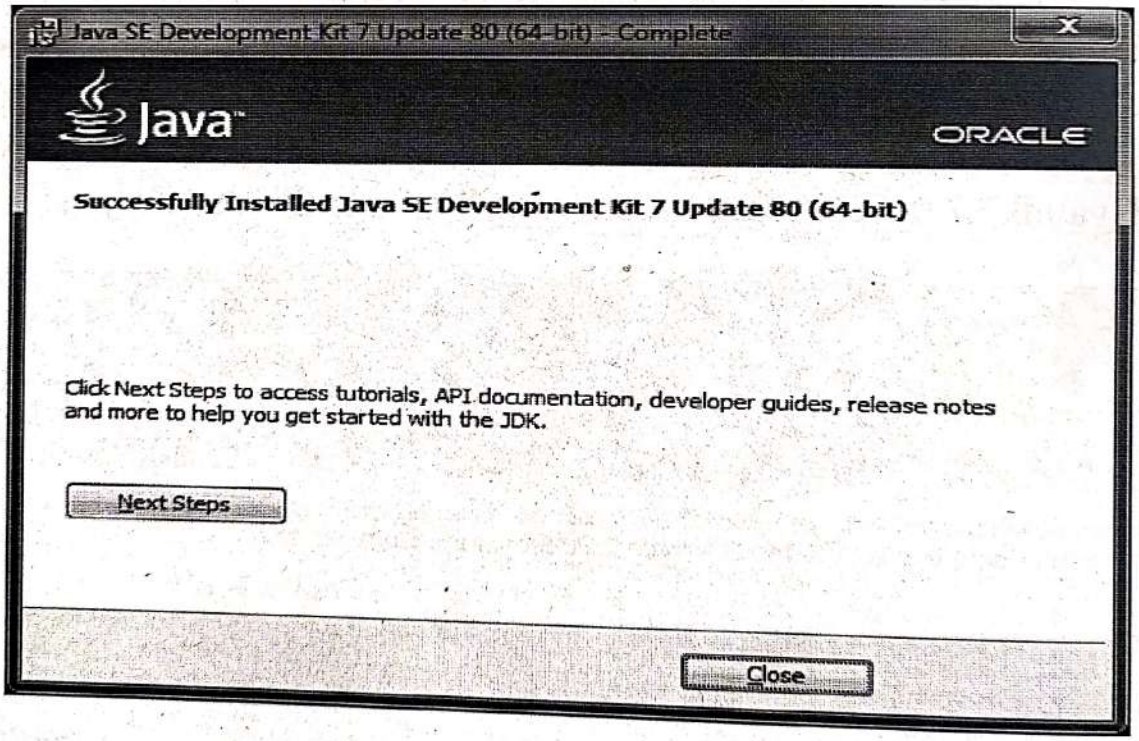
Click Next and on the following screen optionally change the installation location by clicking on the Change Button. In this example the install location was changed to C:\Java\jdk1.7.0_80. From now on we will refer to this directory as:



Next, the installer will present the installation location of the public JRE. We will skip this part of the installer as the JDK installed in the previous step comes with a private JRE that can run developed code. Just press Cancel and confirm by Clicking Yes in the popup window.



Click Next and then Close to finish installing Java.

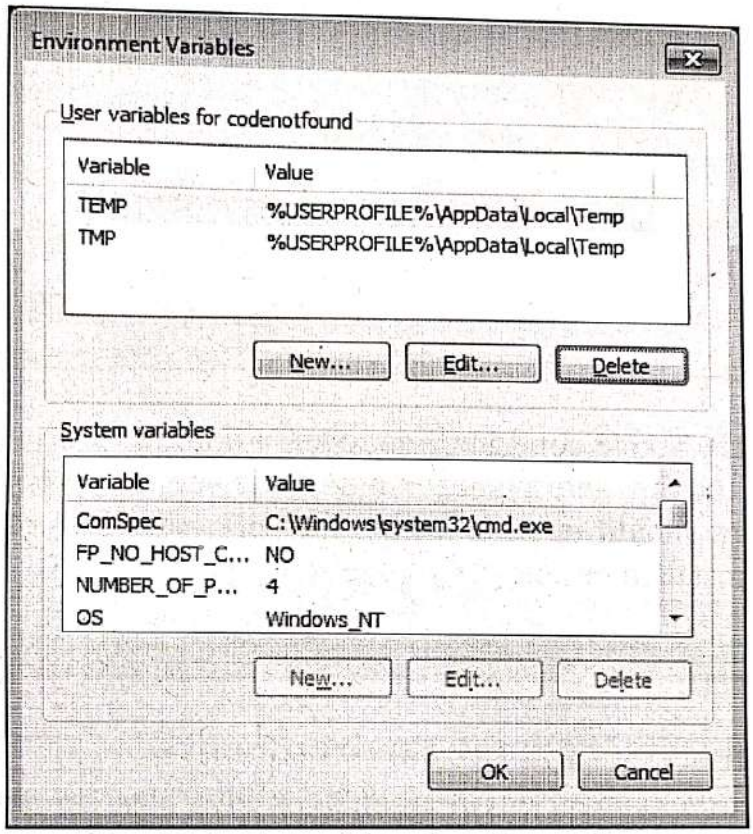


JDK Configuration:

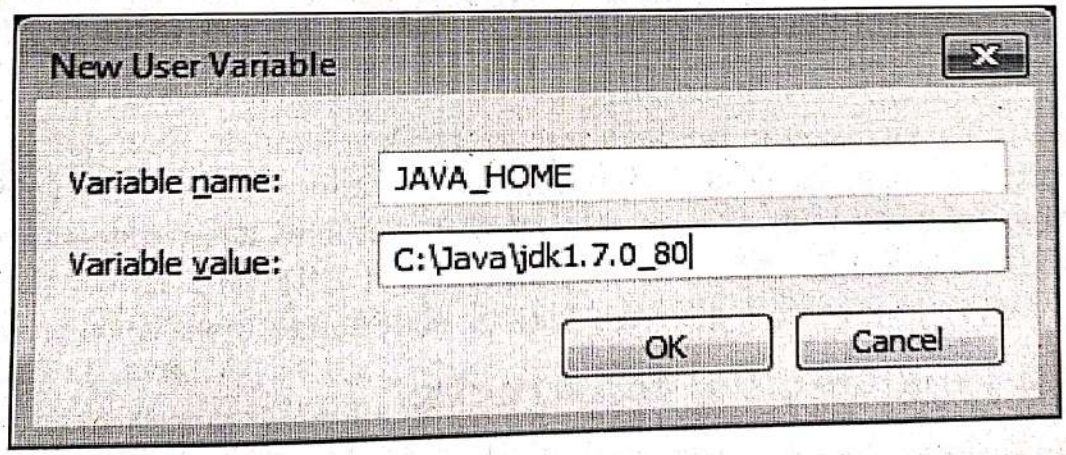
In order for Java applications to be able to run we need to setup a 'JAVA_HOME' environment variable that will point to the Java installation directory. In addition, if we want to run Java commands from a command prompt we need to setup the 'PATH' environment variable to contain the Java bin directory.

When using Windows the above parameters can be configured on the Environment Variables panel. Click on the Windows start button and enter "env".

Environment variables can be set at account level or at system level. For this example click on Edit Environment Variables and following panel should appear.



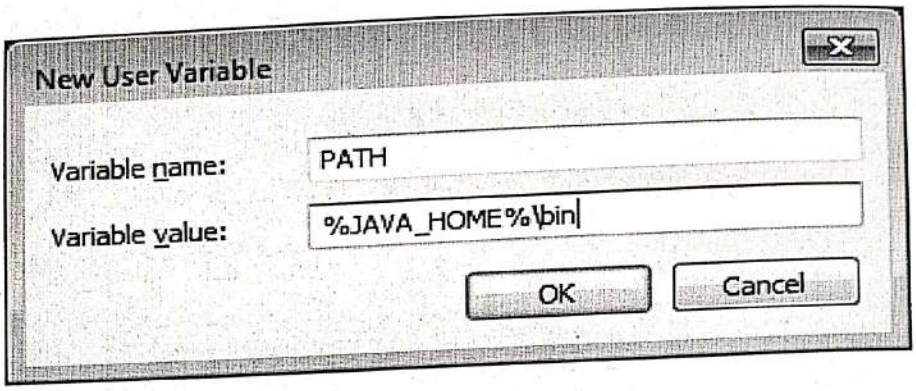
Click on the New button and enter "JAVA_HOME" as variable name and the installation directory as variable value. here the installation directory is 'C:\Java\jdk1.7.0_80'. Click OK to to save.



Click on the New button and enter "PATH" as variable name and "%JAVA_HOME%\bin" as variable value. Click OK to save.

Note that in case a 'Path' variable is already present you can add ";%JAVA_HOME%\bin" at the end of the variable value.

The result should be as shown below. Click OK to close the environment variables panel.



In order to test the above configuration, open a command prompt by clicking on the Windows Start button and typing 'cmd' followed by pressing ENTER. A new command prompt should open in which the following command can be entered to verify that java installed or not:

```
Javac
```

1.1.9 Integrated Development Environment:

Java integrated development environments, or Java ides, are software platforms that provide programmers and developers with a comprehensive set of tools for software development in a single product, specifically in the Java programming language. Java ides are built to work with specific application platforms and remove barriers involved in the lifecycle of software development. Java ides are used by development teams to build new software, apps, web pages, and services, delivering a single tool with all the features needed to accomplish these tasks and removing the need for integrations.

To qualify for inclusion in the Java Integrated Development Environments category, Java programming capabilities through a text editor or a GUI (graphical user interface) Integrate with at least one platform without a separate plugin expose a platform's application programming interface (API) and allow for compiling, debugging, version control, platform-specific code suggestions, or code deployment.

1.2 Java Programming constructs:

1.2.1 Variables:

The data, or variables, defined within a **class** are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class. In most classes, the instance variables are acted upon and accessed by the methods defined for that class. Thus, it is the methods that

determine how a class data can be used. Here is a class called **Box** that defines three instance variables: **width**, **height**, and **depth** and Methods Init() and Display().

```
class Box {
        int Width;
        int Height;
        int Depth;

        void init()
        {
                Width = 5;
                Height = 3;
                Depth = 2;
        }
Void Display()
        {
                System.out.println(Width);
                System.out.println(Height);
                System.out.println(Depth);
        }

        public static void main(String args[])
        {
                Box B1= new Box();
                B1.init();
                B1.Display();
        }
}
```

To actually create a **Box** object, you will use a statement like the following:

Box B1 = new Box(); // create a Box object called B1

After this statement executes, **B1** will be an instance of **Box**. Thus, it will have "physical" reality. For the moment, don't worry about the details of this statement. Again, each time you create an instance of a class, you are creating an object that contains its own copy of each instance variable defined by the class.

Thus, every **Box** object will contain its own copies of the instance variables **width**, **height**, and **depth**. To access these variables and methods, you will use the *dot* (.) operator when you create a class, you are creating a new data type. You can use this type to declare objects of that type. However, obtaining objects of a class is a two-step process.

First, you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object.

Second, you must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the **new** operator.

The **new** operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it. This reference is, more or less, the address in memory of the object allocated by **new**.

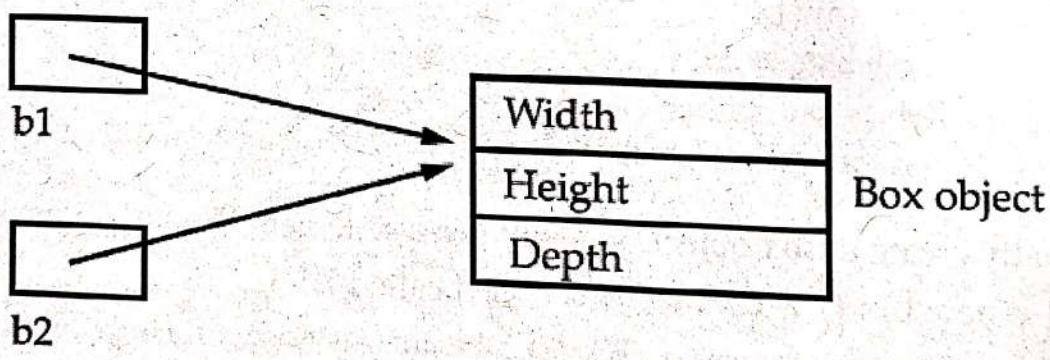
```
Box mybox = new Box();
Or
Box mybox;          // declare reference to object
mybox = new Box();  // allocate a Box object
```

Assigning Object Reference Variables:

Object reference variables act differently than you might expect when an assignment takes place.

```
Box b1 = new Box();
Box b2 = b1;
```

You might think that **b2** is being assigned a reference to a copy of the object referred to by **b1**. That is, you might think that **b1** and **b2** refer to separate and distinct objects. However, this would be wrong. Instead, after this fragment executes, **b1** and **b2** will both refer to the *same* object. The assignment of **b1** to **b2** did not allocate any memory or copy any part of the original object. It simply makes **b2** refer to the same object as does **b1**. Thus, any changes made to the object through **b2** will affect the object to which **b1** is referring, since they are the same object.
This situation is depicted here:



Although **b1** and **b2** both refer to the same object, they are not linked in any other way. For example, a subsequent assignment to **b1** will simply *unhook* **b1** from the original object without affecting the object or affecting **b2**.

For example:

```
Box b1 = new Box();
Box b2 = b1;
// ...
b1 = null;
```

Here, **b1** has been set to **null**, but **b2** still points to the original object.

1.2.2 Primitive Data Types:

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

There are 8 types of primitive data types:

- boolean - Default default value 'false' and size 1 bit
- byte - Default value 0 and size 1 byte
- char - Default store a single character or ASCII value and size 2 byte
- short - Default value 0 and size is 2 byte
- int – Default value 0 and size is 4 byte
- long – Default value 0L and size is 8 byte
- float – Default value 0.0f and size is 4 byte
- double – Default value 0.0d and size is 8 byte

➢ **Boolean:**

The Boolean data type is used to store only two possible values: true and false.

➢ **Byte:**

The byte data type can store whole numbers from -128 to 127. This can be used instead integer types to save memory when you are certain that the value will be within -128 and 127.

```
byte anynum =100;
System.out.println(anynum);
```

➢ **Char:**

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' . The char data type is used to store characters.

```
char nchar = 'j';
System.out.println(nchar);
```

➢ **Short:**

The short data type can store whole numbers from -32768 to 32767.

```
short anynum =1000;
System.out.println(anynum);
```

➢ **Int:**

The int data type can store whole numbers from -2147483648 to 2147483647. The int data type is the preferred data type when we create variables with a numeric value.

```
int anynum =100;
```

```
System.out.println(anynum);
```

- **Long:**

The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807.long data type is used when int is not large enough to store the value. Note that you should end the value with an "L".

```
long anynum =2500000000000L;
System.out.println(anynum);
```

- **Float:**

The float data type can store fractional numbers from 3.4e−038 to 3.4e+038. Note that you should end the value with an "f".

```
float anynum = 2.25f;
System.out.println(anynum);
```

- **Double:**

The double data type is generally used for decimal values just like float. Its value range is unlimited.It is double-precision floating point value. Note that you should end the value with a "d":

```
double anynum = 20.88d;
System.out.println(anynum);
```

1.2.3 Identifier:

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. we cannot use reserve word as identifier in java.

Identifier cannot be a keyword. They are case-sensitive.
It can have a sequence of letters and digits. Identifiers should not start with digits. We are not use special character or symbols. i.e "hello@" is not valid. Whitespaces are not allowed.

In the HelloWorld program given below,
HelloWorld(classname), String(predefinedclassname), args(variable name), main and println(method) all are identifiers.

```
class HelloWorld
{
    Public static void main(String args[])
    {
        System.out.println("HelloWorld");
    }
}
```

1.2.4 Literals:

Any constant value which can be assigned to the variable is called as literal/constant. It can be anything like number, text which represents a value.

For example,

```
int x = 100;
```

So, here 100 is literal.
There are 5 types of Literals in Java.

1. **Integral Literals:**

We can specify the integer literals in 4 different ways –
Decimal (Base 10) :- Digits from 0-9 are allowed in this form.

```
int x = 101;
```

Octal (Base 8):- Digits from 0 – 7 are allowed. It should always have a prefix 0.

```
int x= 0123;
```

Hexa-Decimal (Base 16):-Digits 0-9 are allowed and also characters from a-f are allowed in this form. Furthermore, both uppercase and lowercase characters can be used.

```
int x = 0X123Hello;
```

Binary:-A literal in this type should have a prefix 0b and 0B, i.e. 0 and 1

```
int x = 0b1112;
```

2. **Floating-Point Literals:**

Here, datatypes can only be specified in decimal forms only.
Decimal (Base 10):- Every floating type is a double type and this reason why we cannot assign it directly to float variable, to escape this situation we use f or F as suffix, and for double we use d or D.

3. **Char Literals:**

These are the four types of char:
Single Quote:- Java Literal can be specified to a char data type as a single character within a single quote.

```
char ch = 'a';
```

Char as Integral:-
A char literal in Java can specify as integral literal which also represents the Unicode value of a character. Furthermore, an integer can specify in decimal, octal and even hexadecimal type, but the range is 0-65535.

```
char ch = 062;
```

Unicode Representation:- Char literals can specify in Unicode representation '\uxxxx'.

```
char ch = '\u0061';
```

Escape Sequence:- Escape sequences can also specify as char literal.

```
char ch = '\n';
```

4. String Literals:

String literals are any sequence of characters with a double quote.

```
String s = "Hello";
```

5. Boolean Literals:

They allow only two values i.e. true and false.

```
boolean b = true;
```

1.2.5 Operators:

An operator is a character that **represents an action**, for example + is an arithmetic operator that represents addition.("+ is also use for concatenation of string in java").

Types of Operator in Java:

1) Arithmetic Operators
2) Assignment Operators
3) Auto-increment and Auto-decrement Operators
4) Logical Operators
5) Relational Operators
6) Bitwise Operators
7) Ternary Operator

1) Arithmetic Operators:

Arithmetic operators are: +, -, *, /, %.
where + is for addition of two number.
– is subtraction from one number to other.
* is for multiplication of two number.
/ is for division of two number.
% is to find modulo of a number.

```
public class atithmetic_demo{
public static void main(String args[]){
                int i=20,j=10;
                System.out.println("Addition of Two Number="+(i+j));
                System.out.println("Substract one number from other="+(i-j));
                System.out.println("Multiplication of Two Number="+(i*j));
                System.out.println("Division of Two Number="+(i/j));
```

```
            System.out.println("Modulo of the Number="+(i%j));
        }
    }
```

2) Assignment Operators:

Assignments operators in java are: =, +=, -=, *=, /=, %=
The assignment operator assigns the value on its right to the variable on its left.
Here, 5 is assigned to the variable using = operator.

```
public class AssignmentOperator {
  public static void main(String[] args) {
      int number1, number2;

      // Assigning 5 to number1
      number1 = 5;
      System.out.println(number1);

      // Assigning value of variable number2 to number1
      number2 = number1;
      System.out.println(number2);
  }
}
```

number2+=number1 is equal to number2 = number2+number1
number2-=number1 is equal to number2 = number2-number1
number2*=number1 is equal to number2 = number2*number1
number2/=number1 is equal to number2 = number2/number1
number2%=number1 is equal to number2 = number2%number1

3) Auto-increment and Auto-decrement Operators:

++ and —
number++ is equivalent to number=number+1;
number— is equivalent to number=number-1;

4) Logical Operators:

Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition.
They are &&, ||, !

Let's say we have two boolean variables number1 and number2.
number1**&&** number2 will return true if both number1 and number2 are true else it would return false.

number1|| number2 will return false if both number1 and number2 are false else it would return true.

!number1 would return the opposite of number1, that means it would be true if number1 is false and it would return false if number1 is true.

5) Relational Operators:

We have relational operators in java are : ==, !=, >, <, >=, <=

== returns true if both the left side and right side are equal
!= returns true if left side is not equal to the right side of operator.
> returns true if left side is greater than right.
< returns true if left side is less than right side.
>= returns true if left side is greater than or equal to right side.
<= returns true if left side is less than or equal to right side.

```
class Relational_op
{
public static void main(String args[])
{
float a=15.0f,b=20.75f,c=15.0f;
System.out.println("a="+a);
System.out.println("b="+b);
System.out.println("c="+c);
System.out.println("a<b is="+(a<b));
System.out.println("a>b is="+(a>b));
System.out.println("a==c is="+(a==c));
System.out.println("a<=c is ="+(a<=c));
System.out.println("a>=b is ="+(a>=b));
System.out.println("b!=c is="+(b!=c));
}
}
```

6) Bitwise Operators:

There are six bitwise Operators: &, |, ^, ~, <<, >>

```
number1=11;/*equal to 00001011*/
number2 = 22; /* equal to 00010110 */
```

Bitwise operator performs bit by bit processing.

number1 & number2 compares corresponding bits of number1 and number2 and generates 1 if both bits are equal, else it returns 0. In our case it would return: 2 which is 00000010 because in the binary form of num1 and num2 only second last bits are matching.

number1 | number2 compares corresponding bits of both numbers and generates 1 if either bit is 1, else it returns 0. In our case it would return 31 which is 00011111

number1 ^ number2 compares corresponding bits of both numbers and generates 1 if they are not equal, else it returns 0. In our example it would return 29 which is equivalent to 00011101

~number1 is a complement operator that just changes the bit from 0 to 1 and 1 to 0. In our example it would return -12 which is signed 8 bit equivalent to 11110100

number1 << 2 is left shift operator that moves the bits to the left, discards the far left bit, and assigns the rightmost bit a value of 0. In our case output is 44 which is equivalent to 00101100

number1 >> 2 is right shift operator that moves the bits to the right, discards the far right bit, and assigns the leftmost bit a value of 0. In our case output is 2 which is equivalent to 00000010.

7) **Ternary Operator:**

This operator evaluates a Boolean expression and assigns the value based on the result.

number1 = (expression)? Value if true: Value if false If the expression results true then the first value before the colon (:) is assigned to the variable number1 else the second value is assigned to the number1.

1.2.6 Expressions:

Expressions are essential building blocks of any Java program, usually created to produce a new value, although sometimes an expression assigns a value to a variable. Expressions are built using values, variables, operators, literals and method calls.

Example:-

```
int height;
height = 30;
```

Here, height is an expression that returns int.

```
Double a =1.2,b=1.5,result;
result = a+b;
```

Here, a+b is an expression.

```
if(a==b)
System.out.println("a is equal to b");
```

Here, a==b is an expression that returns Boolean. Similarly, "a is equal to b" is a string expression.

1.2.7 Precedence Rules and Associativity:

Java operators have two properties those are precedence, and *associativity*. Precedence is the priority order of an operator, if there are two or more operators in an expression then the operator of highest priority will be executed first then higher, and then high.

For example, in expression 1 + 2 * 5, multiplication (*) operator will be processed first and then addition. It's because multiplication has higher priority or precedence than addition.

Alternatively, you can say that when an operand is shared by two operators (2 in above example is shared by + and *) then higher priority operator picks the shared operand for processing.

From above example you would have understood the role of precedence or priority in execution of operators. But, the situation may not be as straightforward every time as it is shown in above example.

What if all operators in an expression have same priority? In that case the second property associated with an operator comes into play, which is associativity. Associativity tells the direction of execution of operators that can be either left to right or right to left.

For example, in expression a = b = c =8 the assignment operator is executed from right to left that means c will be assigned by 8, then b will be assigned by c, and finally a will be assigned by b.

parenthesize this expression as (a=(b=(c=8))).

Precedence	Operator	Description	Associativity
1	[] () .	array index method call member access	Left -> Right
2	++ -- + - ~ !	pre or postfix increment pre or postfix decrement unary plus, minus bitwise NOT logical NOT	Right -> Left
3	new	object creation	Right -> Left
4	* / %	multiplication division modulus (remainder)	Left -> Right
5	+ or - +	addition, subtraction string concatenation	Left -> Right
6	<< >> >>>	left shift right shift unsigned or zero-fill right shift	Left -> Right

7	< <= > >= instanceof	less than less than or equal to greater than greater than or equal to reference test	Left -> Right
8	== !=	equal to not equal to	Left -> Right
9	&	bitwise AND	Left -> Right
10	^	bitwise XOR	Left -> Right
11	|	bitwise OR	Left -> Right
12	&&	logical AND	Left -> Right
13	||	logical OR	Left -> Right
14	? :	conditional (ternary)	Right -> Left

1.2.8 Primitive type conversion and casting:

In Java, there are two kinds of data types – primitives and nonprimitive. Primitives include int, float, long, double etc. Non primitive can be of type classes, interfaces, arrays.

In some cases changes do not occur on its own, the programmer needs to specify the type explicitly. This is called **casting**.

Implicit and Explicit Changes:

There are situations in which the system itself changes the type of an expression implicitly based on some requirement.

```
int number1 = 4;
double number2;
double d = number1 / number2;  //number1 is also converted to double
```

This kind of conversion is called **automatic type conversion**.

1.2.9 Flow of Control:

In Java language there are several keywords that are used to alter the flow of the program. Statements can be executed multiple times or only under a specific condition.

The if, else, and switch statements are used for testing conditions, the while and for are looping statements, and the break and continue statements to alter a loop.

When the program is run, the statements are executed from the top to the bottom. One by one.

if statement:

The basic format of an if statement is as follows:

```
if (booleanExpression) {
System.out.println("inside if statement");
}
```

The expression in parentheses must evaluate to (a boolean) true or false. Typically you are test to run something to see if it is true, and then running a code block (one or more statements) if it is true, and (optionally) another block of code if it is not.

The following code demonstrates a legal if-else statement:

```
if (x > 3) {
System.out.println("x is greater than 3");
} else {
System.out.println ("x is not greater than 3");
}
```

The else block is optional.

you can also use the following:

```
if (x >3) {
y = 2;
}
z += 8;
a = y + x;
```

The preceding code will assign 2 to y if the test succeeds (meaning x really is greater than 3), but the other two lines will execute regardless. Even the curly braces are optional if you have only one statement to execute within the body of the conditional block.

The following code example is legal (although not recommended for readability):

```
if (x>3) // bad practice
y = 2;
z += 8;
a = y + x;
```

Sun considers it good practice to enclose blocks within curly braces, even if there is only one statement in the block. Be careful with code like the above, because

You might have a need to nest if-else statements . You can set up an if-else statement to test for multiple conditions.

The following example uses two conditions so that if the first test fails, we want to perform a second test before deciding what to do:

```
if (price < 300) {
buy();
```

```
} else if (price < 400)  {
getApproval();
}
else {
dontBuy();
}
```

Switch Statement:

A way to simulate the use of multiple if statements is with the switch statement. Switch statement is a selection control flow statement.Each branch ends with break keyword.

Take a look at the following if-else code, and notice how confusing it can be to have nested if.

```
int x = 3;
if(x == 1) {
System.out.println("x equals 1");
}
else if(x == 2) {
System.out.println("x equals 2");
}
else if(x == 3) {
System.out.println("x equals 3");
}
else {
System.out.println("No idea what is the value of x");
}
```

Now let's see the same functionality represented in a switch construct:

```
int x = 3;
switch (x) {
case 1:
System.out.println("x equals 1");
break;
case 2:
System.out.println("x equals 2");
break;
case 3:
System.out.println("x equals 3");
break;
default:
System.out.println("No idea what is the value of x");
}
```

Legal Expressions for switch and case:

The general form of the switch statement is:

```
switch (expression) {
case constant1: code block
case constant2: code block
default: code block
}
```

A switch expression must evaluate to a char, byte, short, int, or, as of Java 6, an enum. That means if you are not using an enum, only variables and values that can be automatically promoted (in other words, implicitly cast) to an int are acceptable. You won't be able to compile if you use anything else, including the remaining numeric types of long, float, and double.

A case constant must evaluate to the same type as the switch expression can use, with one additional—and big—constraint: the case constant must be a compile time constant! Since the case argument has to be resolved at compile time, that means you can use only a constant or final variable that is assigned a literal value.

It is not enough to be final, it must be a compile time constant.

For example:

```
final int a = 1;
final int b;
b = 2; int x = 0;
switch (x) {
case a: // ok
case b: // compiler error
```

Also, the switch can only check for equality. This means that the other relational operators such as greater than are rendered unusable in a case. The following is an example of a valid expression using a method invocation in a switch statement.

Note that for this code to be legal, the method being invoked on the object reference must return a value compatible with an int.

```
String s = "xyz";
switch (s.length()) {
case 1:
System.out.println("length is 1");
break;
case 2:
System.out.println("length is 2");
break;
case 3:
System.out.println("length is 3");
break;
default:
```

```
System.out.println("String length not match");
}
```

Using while Loop:

The while loop is good for that type of situation where you do not know how many times a block or statement should repeat, but you want to continue looping as long as some condition is true. A while statement looks like this:

```
while (expression) {
// do stuff
}
Or
int x = 2;
while(x == 2) {
System.out.println(x);
++x;
}
```

In this case, as in all loops, the expression (test) must evaluate to a Boolean result. The body of the while loop will only execute if the expression (condition) results in a value of true. Once inside the loop, the loop body will repeat until the condition is no longer met because it evaluates to false.

In the previous example, program control will enter the loop body because x is equal to 2. However, x is incremented in the loop, so when the condition is checked again it will evaluate to false and exit the loop.

Any variables used in the expression of a while loop must be declared before the expression is evaluated. In other words, you cannot say

```
while (int x = 2) { } // not legal
```

The key point to remember about a while loop is that it might not ever run. If the test expression is false the first time the while expression is checked, the loop body will be skipped and the program will begin executing at the first statement after the while loop. Look at the following example:

```
int x = 8;
while (x > 8) {
System.out.println("in the loop");
x = 10;
}
System.out.println("past the loop");
```

Running this code produces past the loop Because the expression (x > 8) evaluates to false, none of the code within the while loop ever executes.

Using do Loops:

The do loop is similar to the while loop, except that the expression is not evaluated until after the do loop code is executed. Therefore the code in a do loop is guaranteed to execute at least once.
The following shows a do loop in action:

```
do {
System.out.println("inside the loop");
} while(false);
```

The System.out.println() statement will print once, even though the expression evaluates to false. Remember, the do loop will always run the code in the loop body at least once. Be sure to note the use of the semicolon at the end of the while expression.

Using for loop:

Loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

Syntax of for loop:

```
for(initialization ;condition ;increment/decrement)
{
        Statement to be executed();
}
```

In for loop, initialization happens first and only one time, which means that the initialization part of for loop only executes once.

Condition in for loop is evaluated on each iteration, if the condition is true then the statements inside for loop body gets executed. Once the condition returns false, the statements in for loop does not execute and the control gets transferred to the next statement in the program after for loop.

After every execution of for loop's body, the increment/decrement part of for loop executes that updates the **counter.**

After third step, the control jumps to second step and condition is re-evaluated.

```
public class ForLoopExample {
  public static void main(String args[]){
    for(int i=1; i<=10; i++){
        System.out.println("The value of i is: "+i);
    }
  }
}
```

Using break and continue:

The break and continue keywords are used to stop either the entire loop (break) or just the current loop (continue). Typically if are using break or continue, youw will do an if test within the loop, and if some condition becomes true (or false depending on the program), you want to get out immediately.

The difference between them is whether or not you continue with a new iteration or jump to the first statement below the loop and continue from there.

The break statement causes the program to stop execution of the inner most loop and start processing the next line of code after the block.

The continue statement causes only the current iteration of the innermost loop to cease and the next iteration of the same loop to start if the condition of the loop is met.

When using a continue statement with a for loop, you need to consider the effects that continue has on the loop iteration.

Examine the following code:

```
for (int i = 0; i < 10; i++) {
System.out.println("inside loop");
if (B1.doSomething() == 5) {
continue;
}
// more loop code, that won't be reached when the above if
// test is true
}
```

1.3 Classes and Objects (Class, Objects, Class declaration in java, Creating Objects and Methods):

- **Simple Class:**

A class is declared by use of the **class** keyword. The classes that have been used up to this point are actually very limited examples of its complete form. Classes can (and usually do) get much more complex. The general form of a **class** definition is shown here:

```
class classname
{
        type instance-variable1;
        type instance-variable2;
        // ...
        type instance-variableN;
```

```
        type methodname1(parameter-list)
        {
                // body of method
        }
        type methodname2(parameter-list)
        {
                // body of method
        }
}
```

➢ **Constructors:**

Java allows class to create special member function that automatically calls itself when object of that particular class is created. That function is called constructor. Constructor carries same name as the class name. This characteristics differentiate it from other normal function. That means when you found a function with the same name as class name that function is constructor.

Types of Java constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Characteristics of Constructor:

1. It contain same name as class name.
2. It is called automatically when object of a particular class is created.
3. It contain no return statement and no return type not even void.
4. It cannot be invoked or called explicitly or manually.
5. Java constructor cannot be abstract, static, final, and synchronized

Example :

```
/* Here, Box uses a constructor to initialize the
dimensions of a box.
*/
class Box
{
        double width;
        double height;
        double depth;

        // This is the constructor for Box.
        Box()
        {
                System.out.println("Constructing Box");
```

```
            width = 10;
            height = 10;
            depth = 10;
        }
        double volume()
        {
            return width * height * depth;
        }
}

class BoxDemo6
{
    public static void main(String args[])
    {
        // declare, allocate, and initialize Box objects
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;
        // get volume of first box
        vol = mybox1.volume();
        System.out.println("Volume is " + vol);
        // get volume of second box
        vol = mybox2.volume();
        System.out.println("Volume is " + vol);
    }
}
```

➢ **The this Keyword:**

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the **this** keyword. **this** can be used inside any method to refer to the current object. That is, **this** is always a reference to the object on which the method was invoked. You can use **this** anywhere a reference to an object of the current class' type is permitted. To better understand what **this** refers to, consider the following version of **Box( )**:

```
Box(double width, double height, double depth)
{
    this.width = width;
    this.height = height;
    this.depth = depth;
}
```

This version of **Box( )** operates exactly like the earlier version. The use of **this** is redundant, but perfectly correct. Inside **Box( )**, **this** will always refer to the invoking object. While it is redundant in this case, **this** is useful in other contexts, one of which is explained in the next section.

➢ **Cleaning Up Unused Objects:**

Since objects are dynamically allocated by using the **new** operator, you might be wondering how such objects are destroyed and their memory released for later reallocation. In some languages, such as C++, dynamically allocated objects must be manually released by use of a **delete** operator. Java takes a different approach; it handles de-allocation for you automatically. The technique that accomplishes this is called garbage collection. It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no explicit need to destroy objects as in C++. Garbage collection only occurs during the execution of your program. It will not occur simply because one or more objects exist that are no longer used. Furthermore, different Java run-time implementations will take varying approaches to garbage collection, but for the most part, you should not have to think about it while writing your programs.

➢ **The finalize( ) Method (Destructor):**

Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called *finalization*. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector. To add a finalizer to a class, you simply define the **finalize( )** method. The Java run time 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 garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed, the Java run time calls the **finalize( )** method on the object.

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

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

Here, the keyword **protected** is a specifier that prevents access to **finalize( )** by code defined outside its class. It is important to understand that **finalize( )** is only called just prior to garbage collection. It is not called when an object goes out-of-scope, for example. This means
that you cannot know when—or even if—**finalize( )** will be executed. Therefore, your program should provide other means of releasing system resources, etc., used by the object. It must not rely on **finalize( )** for normal program operation.

Exercises

Answer the following Question (1 Mark each)

1) What is class?
2) What do you mean by an object?
3) What is inheritance?
4) What is Encaptulation?
5) What is the functionality of JVM?
6) Can we execute code before main() method?
7) Java doesn't use pointers. Why?
8) How many different datatypes used in java?
9) What do you mean by Constructor?
10) What is meant by Local variable and Instance variable?

State which of the following are True or False.

1) Java is a one platform.
2) Java is programming language means it use for web only.
3) The size of char datatype is 1 byte.
4) The size of data type is platform dependent.
5) goto can be used to jump to a statement in Java.
6) The break statement is required in the default case of a switch selection structure.

7) String is primitive data type.
8) A single Java file can contain multiple classes.
9) Object is an instance of a class.
10) The size of char data-type is 2 byte in java.
11) The default case is required in the switch selection structure.
12) Java source code can be written in files with any extension
13) One class cannot be derived from more than one class in java.

Answer the following Questions (5 marks each)

1) Explain the features of JAVA.
2) Explain all the concept of object oriented programming language.
3) Discuss the concept of classes and objects with respect to object oriented programming
4) Explain all primitive data types.
5) Explain the use of break and continue statements.
6) Explain the arithmetic and relational operators using suitable examples
7) Discuss the different types of loop statements using suitable examples
8) Explain importance of Constructor.

Multiple choice Question-Answer.

1) Which of the following are legal lines of Java code?

```
Int w =(int)888.8;
Byte x = (byte)100L;
Long y = (byte)100;
Byte z = (byte)100L;
```

a) 1 and 2
b) 2 and 3
c) 3 and 4
d) All statements are correct

Answer :- d

2) If an expression contains double, int, float, long, then the whole expression will be promoted into which of these data types?
a) long
b) int
c) double
d) float

Answer :-c

3) Which of the following can be operands of arithmetic operators?
a) Numeric
b) Boolean
c) Characters
d) Both Numeric & Characters

Answer :- d

4) What should be expression1 evaluate to in using ternary operator as in this line?
expression1 ? expression2 : expression3
a) Integer
b) Floating – point numbers
c) Boolean
d) None of the mentioned
Answer:- c

5) Which of these selection statements only for equality?
a) if
b) switch
c) if & switch
d) none of the mentioned
Answer :- b

6) What is true about a break?
a) Break stops the execution of entire program
b) Break halts the execution and forces the control out of the loop
c) Break forces the control out of the loop and starts the execution of next iteration
d) Break halts the execution of the loop for certain time frame
Answer:-b

7) Which of the following is not OOPS concept in Java?
a) Inheritance
b) Encapsulation
c) Polymorphism
d) Compilation
Answer :- d

9) Which component is responsible to optimize bytecode to machine code?
a) JVM
b) JDK
c) JIT
d) JRE
Answer:-c

10) What is the return type of Constructors?
a) int
b) float
c) void
d) none of the mentioned
Answer :- d

11) Which function is used to perform some action when the object is to be destroyed?
a) finalize()
b) delete()
c) main()
d) none of the mentioned
Answer:-a

☺ ☺ ☺ ☺ ☺

CC-214 Core Java Practical (UNIT – 1)

1. Write a program to evaluate simple interest of a given principal, rate and time.

```
public class Main
{
  public static void main (String args[])
   {  float p, r,  t,  si;
// principal amount, rate, time and simple interest respectively
        p = 13000;  r = 12; t = 2;
        si  = (p*r*t)/100;
        System.out.println("Simple Interest is: " +si);
  }
}
```

Output:
Simple Interest is: 3120.0

2. A motor cycle dealer sells two wheelers to his customrer on loa which is to be repaid in 5 yyears. the dealer charges simp interest for the whle term on the day of giving the loan itself. th total ammount is then divided by 60(months) and is collectd a equated monthly instalment (emi) . write a program to calcula the emi for a loan of Rs. X where X is given form command li argument. Print the EMI value in Ruppes.

```
import javax.swing.JOptionPane;
class Loan
{
int loan;
int interest;
double emi;
Loan(int loan,int interest)
{
        this.loan=loan;
        this.interest=interest;
}
double calcEMI()
        {
```

```
            return loan*interest/(100*60);
        }
}
class U1P2
{
        public static void main(String arg[])
        {
                int loan;
                int interest;
                String lo=JOptionPane.showInputDialog(null,"Enter Loan
        Ammount");
                loan=Integer.parseInt(lo);
                String in=JOptionPane.showInputDialog(null,"Enter Interest");
                interest=Integer.parseInt(in);
                Loan l1=new Loan(loan,interest);
                System.out.println("EMI="+l1.calcEMI());
        }
}
```

Output:

Enter Loan Ammount: 1000
Enter Interest: 10
EMI=1.0

3. A car accessories shop assigns code 1 to seat covers, 2 to steering wheel covers , 3 to car lighting and 4 for air purifiers. All other items have code 5 or more. While selling the goods, a sales tax of 2% to seat covers ,3% to steering wheel covers, 4% to car lighting, 2.5% to air purifiers and 1.2% for all other items is charged. A list containing the product code and price is given for making a bill. Write a java program using switch statements to prepare a bill.

```
import javax.swing.JOptionPane;
class CarAcc
{
        static int prodPrice[][]={ {1,100},
                {2,200},
                {3,300},
                {4,400},
                {5,500},
```

```
                {6,600} };
        static double salesTax[]={2,3,4,2.5,1.2};
        CarAcc()
        {}
        void getBill(int prCode, int qty)
        {
                double billAmm;
                switch(prCode)
                {
                case 1:
                        billAmm=(prodPrice[prCode-1][1]*qty*salesTax[prCode-1
/100)+ prodPrice[prCode-1][1]*qty;
                        JOptionPane.showMessageDialog(null,"Price="+billAmm);
                break;
                case 2:
                        billAmm=(prodPrice[prCode-1][1]*qty*salesTax[prCode-1
/100)+ prodPrice[prCode-1][1]*qty;
                        JOptionPane.showMessageDialog(null,"Price="+billAmm);
                break;
                case 3:
                        billAmm=(prodPrice[prCode-1][1]*qty*salesTax[prCode-
1]/100)+prodPrice[prCode-1][1]*qty;
                        JOptionPane.showMessageDialog(null,"Price="+billAmm);
                break;
                case 4:
                        billAmm=(prodPrice[prCode-1][1]*qty*salesTax[prCode-
1]/100)+prodPrice[prCode-1][1]*qty;
                        JOptionPane.showMessageDialog(null,"Price="+billAmm);
                break;
                case 5:
                        billAmm=(prodPrice[prCode-
1][1]*qty*salesTax[4]/100)+prodPrice[prCode-1][1]*qty;
                        JOptionPane.showMessageDialog(null,"Price="+billAmm);
                break;
                default: System.out.println("Please enter proper value");
                }
        }
}
class U1P3
{
        public static void main(String arg[])
```

```
        {
                int prCode, qty;
                CarAcc b1=new CarAcc();
                String a=JOptionPane.showInputDialog(null,"Enter Prod Code:");
                prCode=Integer.parseInt(a);
                String b=JOptionPane.showInputDialog(null,"Enter Quantity:");
                qty=Integer.parseInt(b);
                b1.getBill(prCode,qty);
        }
}
```

Output:

Enter Prod Code: 1
Enter Quantity: 200
Price: 20400

4. Write a java Program to scan 3 integer values from the command line argument and display the maximum number using conditional operator.

```
class U1P4
{
public static void main(String arg[])
{
int a, b, c, max;
a=Integer.parseInt(arg[0]);
b=Integer.parseInt(arg[1]);
c=Integer.parseInt(arg[2]);
if(a>b)
{
if(a>c)
max=a;
else
max=c;
}
else
{
if(b>c)
max=b;
else
```

```
max=c;
}
System.out.println("Max among "+a+", "+b+" and "+c+" is .."+max);
}
}
```

Output:

java U1P4 4 2 3
Max among 4, 2 and 3 is ..4

5. Write a java Program to caluclate the hypotenuse of right angled triangle when other sides of the trangle are given. (Hypotenuse= square root (x*x + y*y) using command line argument.

```
class U1P5
{
public static void main(String arg[])
{
int x, y, z;
double hyp;
x=Integer.parseInt(arg[0]);
y=Integer.parseInt(arg[1]);
z=(x*x)+(y*y);
hyp=Math.sqrt(z);
System.out.println("hypotenuse="+hyp);
}
}
```

Output:

E:\docs\javaex>java U1P5 4 3
hypotenuse=5.0

6. Write a java program to calculate the area of square and rectangl by overloading the area method.

```
class AreaCalc
{
double area(int x)
{
return x*x;
}
```

```
double area(int x, int y)
{
return x*y;
}
}
class U1P6
{
public static void main(String arg[])
{
int x,y;
x=Integer.parseInt(arg[0]);
y=Integer.parseInt(arg[1]);
AreaCalc a=new AreaCalc();
System.out.println("Area of square:"+a.area(x));
System.out.println("Area of rectangle:"+a.area(x,y));
}
}
```

Output:

E:\docs\javaex>java U1P6 4 3
Area of square:16.0
Area of rectangle:12.0

7. Create a complex number class. The class should have a constructor and methods to add, subtract and multiply two complex numbers and to return the real and imaginary parts.

```
class Complex
{
int Real,Imag;
Complex()
{}
Complex(int Real1,int Imag1)
{
Real=Real1;
Imag=Imag1;
}
Complex addComplex(Complex C1,Complex C2)
{
Complex CSum=new Complex();
```

```
CSum.Real=C1.Real+C2.Real;
CSum.Imag=C1.Imag+C2.Imag;
return CSum;
}
Complex subComplex(Complex c1,Complex c2)
{
Complex CSub=new Complex();
CSub.Real=c1.Real-c2.Real;
CSub.Imag=c1.Imag-c2.Imag;
return CSub;
}
Complex mulComplex(Complex C1, Complex C2)
{
Complex CMul=new Complex();
CMul.Real=C1.Real*C2.Real -
C1.Imag*C2.Imag;
CMul.Imag=C1.Real*C2.Imag+C1.Imag*C2.Real;
return CMul;
}
}
class U1P7
{
public static void main(String[] a)
{
Complex C1=new Complex(4,8);
Complex C2=new Complex(5,7);
Complex C3=new Complex();
C3=C3.addComplex(C1,C2);
System.out.println("SUM:" + C3.Real +"+i" + C3.Imag);
C3=C3.mulComplex(C1,C2);
System.out.println("Mul:" + C3.Real +"+i" + C3.Imag);
C3=C3.subComplex(C1,C2);
System.out.println("Sub:" + C3.Real +"+i" + C3.Imag);
}
}
```

Output:

```
E:\docs\javaex>java U1P7
SUM:9+i15
Mul:-36+i68
Sub:-1+i1
```

8. Write a java program to display powers of 2 i.e. 2,4,8,16 etc up to 1024 using bitwise operators.

```
class U1P10
{
public static void main(String args[])
{
int a=1;
for (int i=1;i<=10;i++)
{
a=a<<1;
System.out.println(a);
}
}
}
```

Output:

```
E:\docs\javaex>java U1P8
2
4
8
16
32
64
128
256
512
1024
```

Unit - 2

Array, Inheritance and Interface

- What is an array?
- Inheritance
- Interface

Unit-2 Array, Inheritance and Interface

2.1 What is an array?

An array is a group/collection of one or more values of the same type. it is a memory space allocated that can store multiple values of same data type in contiguous locations. These logically contiguous locations can be accessed with a common name. Complete set of values is called an array while each value is called an element of the array. The elements of the array share the same variable name. Each element of any array has its own unique index number/subscript. Java array is an object which contains elements of a similar data type. Arrays can be two types: one-dimensional and multi-dimensional.

2.1.1 One-dimensional array:

Conceptually you can think of a one-dimensional array as a row, where elements are stored one after another. Here, a single subscript/index is used where each index value refers to an individual array element. Like any other variable, arrays must be declared and created in the computer memory before they are used. Creation of an array involves three steps:

- Declaration of the array
- Creating memory locations
- Initializing values to the array

You can declare an array using one of the following Syntaxes:
datatype[] array_name; // int[] marks;
datatype []array_name; // int []marks;
datatype array_name[]; // int marks[];

You can initialize an array using following Syntax:
array_name=new datatype[size]; // marks=new int[5];
Here, datatype denotes the type of elements in the array, array_name must be a valid identifier and size represents total number of elements an array can hold. Java allows to create arrays using new operator only. Using new, the array name acquires an actual memory address value.

You can combine both the above Syntaxes, to create an array as one of the following ways:

datatype[] array_name=new datatype[]; // int[] marks=new int[5];
datatype []array_name=new datatype[]; // int []marks=new int[5];
datatype array_name[]=new datatype[]; // int marks[]=new int[5];

You can initialize/assign values to an array using the following Syntax:

array_name[index]=value; // marks[0]=80;

You can initialize an array at the time of creation of an array itself as:

datatype array_name[]={list of values}; // int marks[]={80,70,60,50,40};

When you initialize an array by giving it values at the time of creation, it is not required to give the array a size as the size is assigned based on the number of values you place at the time of initialization.

Example: declaration and instantiation of a one-dimensional array

```
class 1DArrayDemo
{
public static void main(String args[])
{
 int marks[]=new int[5];
 marks[0]=80;
 marks[1]=70;
 marks[2]=60;
 marks[3]=50;
 marks[4]=40;
 for(int i=0;i<marks.length;i++) // length is the property of array
   System.out.println("marks["+i+"]="+marks[i]);
}
}
```

Output:

```
marks[0]=80
marks[1]=70
marks[2]=60
marks[3]=50
marks[4]=40
```

2.1.2 Two-dimensional array:

Two-dimensional arrays have two or more columns of values. You must use two subscripts when you access an element in a two-dimensional array. For creating two dimensional arrays, we must follow the same steps as that of one-dimensional arrays.

Syntax:

datatype array_name[][]; // int marks[][];

array_name=new datatype[row_size][col_size]; // marks=new int[2][3];

datatype [][] array_name=new int[row_size][col_size]; // int[][] marks=new[2][3]

You can initialize an array at the time of creation of an array itself as:

datatype array_name[][]={list of values};

// int marks[2][3]={80,70,60,85,50,40}; or int marks[][]={(80,70,60),(85,50,40)}; this declaration show that the first two rows (students) and three columns (marks of three subjects).

Example: declaration and instantiation of a two-dimensional array

```
class 2DArrayDemo
{
public static void main(String args[])
{
 int marks[][]=new int[2][3];
 marks[0][0]=80;
 marks[0][1]=70;
 marks[0][2]=60;
 marks[1][0]=85;
 marks[1][1]=50;
 marks[1][2]=40;
 for(int row=0;row<2;row++)
 {
    for(int col=0;col<3;col++)
    {
       System.out.println("marks["+row+"]["+col+"]="+marks[row][col]);
    }
 }
}
}
```

Output:

```
marks[0][0]=80
marks[0][1]=70
marks[0][2]=60
marks[1][0]=85
marks[1][1]=50
marks[1][2]=40
```

2.1.3 Using enhanced for/for...each loop with array:

The enhanced for loop provides a simpler way to iterate through all the elements of an array. This loop allows you to cycle through an array without specifying the starting and ending points for the loop control variable. The limitation of enhanced for loop is that it is possible to iterate in forward direction only by single step.

Syntax:

```
for(datatype loop_control_variable_name:array_name)
{
  //body of the loop
}
```

Example: find sum of elements of an array using for-each loop

```
class ForEachDemo
{
public static void main(String args[])
{
 int marks[]={50,60,70,80,90},sum=0;
 for(int value:marks)
 {
  sum+=value;
 }
 System.out.println("Sum of array elements="+sum);
}
}
```

Output:

Sum of array elements=350

2.1.4 Passing arrays to methods and returning arrays from method:

- As you can pass primitive type (boolean, char, byte, short, int, long, float, double values to methods, you can also pass arrays to a method.
- To pass an array to a method, specify the name of the array without any square brackets within the method call.
- Every array object knows its own length using its length field, therefore while passing array's object reference into a method, we do not need to pass the array length as an additional argument.
- As when you pass a variable of primitive type as an argument to a method, the method actually gets a copy of value stored in the variable. So when an individual array element of primitive type is passed, the matching parameter receives a copy.
- However, when an array is passed as an argument, you just pass the array's reference in matching parameter and not the copy of individual elements. Any modification made in the array using its reference will be visible to the caller.
- In short, when any primitive type is passed to the method, the value is passed. And when you pass an array (non-primitive object) to the method, the reference is passed.
- This means that the object actually holds a memory address where the values are stored and the receiving method gets a copy of the array's actual memory address.
- Therefore, the receiving method has access to and the ability to alter the original values in the array elements in the calling method.

Example: passing an array to the method:

```
class PassArray
 {
public static void main(String[] args)
{
    int[] marks={80,50,90,40,60};
    System.out.println("Array elements before sorting:");
    display(marks);
    sort(marks);
    System.out.println("\nArray elements after sorting:");
    display(marks);
}
static void display(int m[])
{
  for(int i=0; i<m.length;i++)
     System.out.print(m[i] + " ");
}
static void sort(int m[])
{
 int i, j, temp;
 for(i=0; i<m.length-i;i++)
    {
       for(j=0; j<m.length-i-1;j++)
          {
             if(m[j]>m[j+1])
              {
                 temp = m[j];
                 m[j] = m[j+1];
                 m[j+1] = temp;
              }
          }
         }
         }
}
```

Output:

```
Array elements before sorting:
80 50 90 40 60
Array elements after sorting:
40 50 60 80 90
```

Example: returning an array from the method:

```
public class ReturnArray
```

```
{
  public static double[] returnArray( )
  {
    double[] x;
    x = new double[3];
    x[0] = 5.3;
    x[1] = 5.4;
    x[2] = 5.5;
    return(x);
  }
  public static void main(String[] args)
  {
    double[] arr;
    arr = returnArray();
    System.out.println("returning an array from the method:");
    for (int i = 0; i < arr.length; i++)
     System.out.println(arr[i]);
  }
}
```

Output:

returning an array from the method:
5.3
5.4
5.5

2.1.5 Command line arguments:

An argument is passed at the time of running the java program. The arguments passe from the console can be received in the java program and it can be used as an inpu They are stored as string in String array passed to the args paramet of main() method.

Example:

```
class cmd
{
 public static void main(String[] args)
 {
  for(int i=0;i< args.length;i++)
  {
  System.out.println(args[i]);
  }
 }
}
```

Output:

Example: find sum of numbers passed from command line argument

```
public class Add
{
 public static void main(String[] args)
 {
      int sum = 0;
      for (int i = 0; i < args.length; i++)
      {
         sum = sum + Integer.parseInt(args[i]);
      }
      System.out.println("The sum of the arguments passed is " + sum);
 }
}
```

Output:

```
C:\Users\srgf\Desktop>java Add 3 4 7 9 34
The sum of the arguments passed is 57
```

2.2 Inheritance:

The process by which one class acquires the properties (data members) and functionalities (methods) of another class is called inheritance. The aim is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class. The parent class is known as super class and newly created child class is known as the subclass. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

2.2.1 Deriving classes using extends keyword:

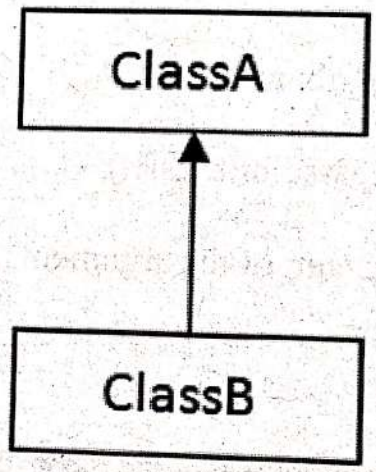
To inherit from a class, use the extends keyword. The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Syntax:

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

Types of Inheritance:

- **Single inheritance:** In single inheritance, classes have only one base class. It refers to a child and parent class relationship where a class extends the another class.



Single

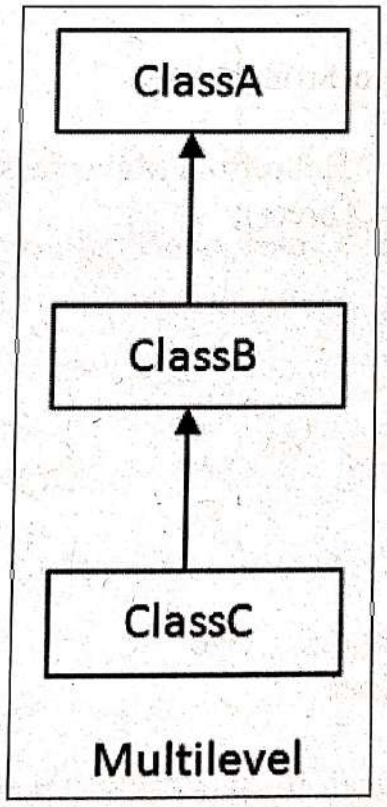
Example: single inheritance

```
class One
{
        public void print_one()
        {
                System.out.println("Hello from Parent class One");
        }
}
class Two extends One
{
        public void print_two()
        {
                System.out.println("Hello from Child class Two");
        }
}
```

```
public class Main
{
        public static void main(String[] args)
        {
                System.out.println("Hello from Main class");
                Two t = new Two();
                t.print_one();
                t.print_two();
        }
}
```

Output:
Hello from Main class
Hello from Parent class One
Hello from Child class Two

- **Multilevel inheritance:** In multilevel inheritance, a class not only inherits from its immediate superclass, but also from its super class. There is no limit to this chain of inheritance but after few levels the complexity of the code will be increased. It refers to a child and parent class relationship where a class extends the child class.



Example: multilevel inheritance

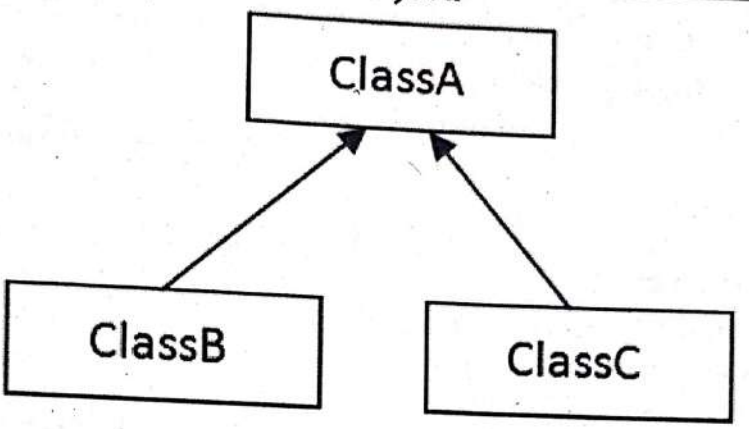
```
class One
{
```

```
        public void print_one()
        {
                System.out.println("Hello from Base class One");
        }
}
class Two extends One
{
        public void print_two()
        {
                System.out.println("Hello from Immediate Parent class Two");
        }
}
class Three extends Two
{
        public void print_three()
        {
                System.out.println("Hello from Derived class Three");
        }
}
public class Main
{
        public static void main(String[] args)
        {
           System.out.println("Hello from Main class");
                Three t = new Three();
                t.print_one();
                t.print_two();
                t.print_three();
        }
}
```

Output:

```
Hello from Main class
Hello from Base class One
Hello from Immediate Parent class Two
Hello from Derived class Three
```

- **Hierarchical inheritance:** In hierarchical inheritance, there is a hierarchies o classes like a tree structure. It refers to a child and parent class relationship where more than one classes extends the same class.



Hierarchical

Example: hierarchical inheritance

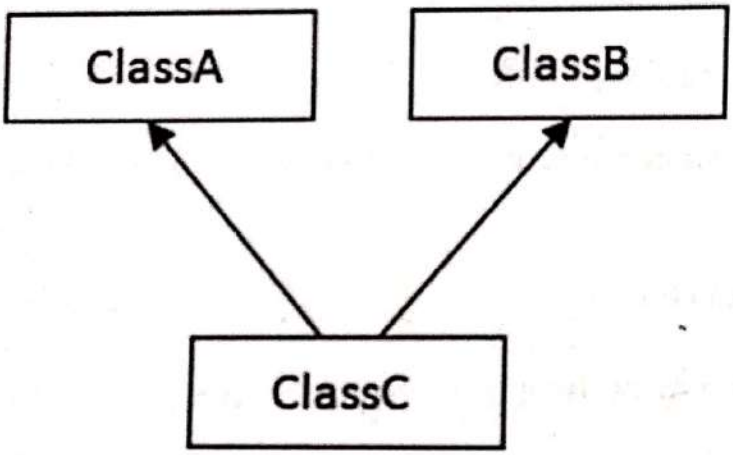
```
class One
{
        public void print_one()
        {
                System.out.println("Hello from Base class One");
        }
}
class Two extends One
{
        public void print_two()
        {
                System.out.println("Hello from Derived class Two");
        }
}
class Three extends One
{
        public void print_three()
        {
                System.out.println("Hello from another Derived class Three");
        }
}
public class Main
{
        public static void main(String[] args)
        {
                System.out.println("Hello from Main class");
                Three th = new Three();
                th.print_three();
```

```
        Two to = new Two();
        to.print_two();
    }
}
```

Output:

Hello from Main class
Hello from another Derived class Three
Hello from Derived class Two

- **Multiple inheritance:** In multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Java does not support multiple inheritance with classes. It can be achieved by the concept of interface.



Multiple

Example:

```
interface One
{
    public void print_one();
}
interface Two
{
    public void print_two();
}
interface Three extends One,Two
{
    public void print_three();
}
```

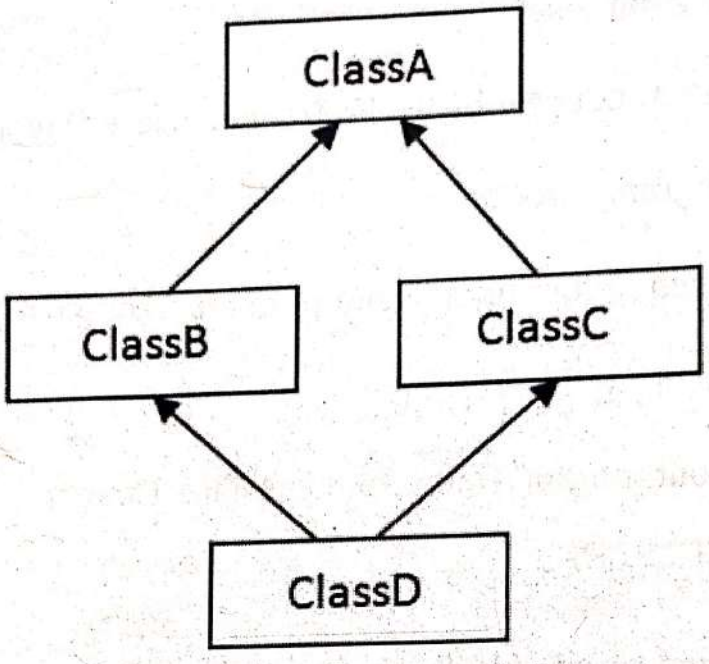
```
class Four implements Three
{
        @Override
        public void print_one()
        {
                System.out.println("Hello from interface One");
        }
        public void print_two()
        {
                System.out.println("Hello from interface Two");
        }
        public void print_three()
        {
           System.out.println("Hello from interface Three");
        }
        public void print_four()
        {
           System.out.println("Hello from class Four");
        }
}
public class Main
{
        public static void main(String[] args)
        {
                System.out.println("Hello from Main class");
                Four f = new Four();
                f.print_one();
                f.print_two();
                f.print_three();
                f.print_four();
        }
}
```

Output:

```
Hello from Main class
Hello from interface One
Hello from interface Two
Hello from interface Three
Hello from class Four
```

- **Hybrid inheritance:** In hybrid inheritance, combination of more than one types of inheritance in a single program. It is a mix of two or more of the above types of

inheritance. Since Java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. One can achieve it through interfaces.



Hybrid

Example: hybrid inheritance

```
class ClassA
{
   public void displayA()
   {
      System.out.println("Display method of ClassA");
   }
}
interface InterfaceB
{
   public void show();
}
interface InterfaceC
{
   public void show();
}
public class ClassD extends ClassA implements InterfaceB,InterfaceC
{
   public void show()
   {
      System.out.println("Show method implementation");
```

```
    }
    public void displayD()
    {
      System.out.println("Display method of ClassD");
    }
    public static void main(String args[])
    {
      ClassD d = new ClassD();
      d.displayD();
      d.show();
    }
}
```

Output:
Display method of ClassD
Show method implementation

2.2.2 Method Overriding:

Declaring a method in sub class which is already present in parent class is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. The method in parent class is called overridden method and the method in child class is called overriding method.

Method Overriding is helpful when a class has several child classes, so if a child class needs to use the parent class method, it can use it and the other classes that want to have different implementation can use overriding feature to make changes without touching the parent class code.

It is also used for runtime polymorphism. When a parent class reference points to the child class object then the call to the overridden method is determined at runtime, because during method call which method (parent class or child class) is to be executed is determined by the type of object.

Keep following things/rules in mind while implementing method overriding.

- The method must have the same name as in the parent class.
- The method must have the same parameter as in the parent class.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- There must be an IS-A relationship (inheritance).
- A method declared final cannot be overridden.
- Constructors cannot be overridden.

Example: method overriding

```
class Vehicle
{
  void run()
  {
     System.out.println("Vehicle is running");
  }
}
class Car extends Vehicle
{
   void run()
   {
      System.out.println("Car is running");
   }
   public static void main(String args[])
   {
      Car obj = new Car();
      obj.run();
   }
}
```

Output:

Car is running

Example: implementing runtime polymorphism

```
class Bank
{
   int getRoI()
   {
      return 0;
   }
}
class IDFC extends Bank
{
   int getRoI()
   {
      return 7;
   }
}
class PayTM extends Bank
{
   int getRoI()
```

```
    {
      return 5;
    }
}
class Bandhan extends Bank
{
   int getRoI()
   {
      return 6;
   }
}
class Main
{
   public static void main(String args[])
   {
      IDFC i=new IDFC();
      PayTM p=new PayTM();
      Bandhan b=new Bandhan();
      System.out.println("IDFC First Bank Rate of Interest: "+i.getRoI());
      System.out.println("PayTM Bank Rate of Interest: "+p.getRoI());
      System.out.println("Bandhan Bank Rate of Interest: "+b.getRoI());
   }
}
```

Output:

IDFC First Bank Rate of Interest: 7
PayTM Bank Rate of Interest: 5
Bandhan Bank Rate of Interest: 6

2.2.3 super keyword:

The super keyword is used for calling the parent class method/constructor. It is used for_

- Calling the methods of the super class.
- Accessing the member variables of the super class.
- Invoking the constructors of the super class.

Example: Calling the methods of the super class

```
class Parentclass
{
   void display()
   {
         System.out.println("Parent class method");
```

```
    }
}
class Subclass extends Parentclass
{
   void display()
   {
         System.out.println("Child class method");
   }
   void printMsg()
   {
         display();
         super.display();
   }
   public static void main(String args[])
   {
         Subclass obj= new Subclass();
         obj.printMsg();
   }
}
```

Output:

Child class method
Parent class method

Example: Accessing the member variables of the super class

```
class Parentclass
{
  int num = 10;
}
class Subclass extends Parentclass
{
  int num = 20;
  void printNum()
  {
    System.out.println(super.num);

  }
  public static void main(String args[])
  {
    Subclass obj= new Subclass();
    obj.printNum();
  }
}
```

Output:

10

Example: Invoking the constructors of the super class

```
class Person
{
  int id;
  String name;
  Person(int id,String name)
  {
    this.id=id;
    this.name=name;
  }
}
class Emp extends Person
{
  float salary;
  Emp(int id,String name,float salary)
  {
    super(id,name);
    this.salary=salary;
  }
  void display()
  {
    System.out.println(id+" "+name+" "+salary);
  }
}
class Main
{
  public static void main(String[] args)
  {
    Emp e1=new Emp(1234,"Dr. Shivang Patel",125000f);
    e1.display();
  }
}
```

Output:

1234 Dr. Shivang Patel 125000.0

Order of execution of constructors:

The order of execution when creating the object of child class is - parent class constructor is executed first and then the child class constructor is executed. It

happens because compiler itself adds super() as the first statement in the constructor of child class.

Example: Implicitly invoking the No-argument constructors of the super class

```
class Parentclass
{
  Parentclass()
  {
        System.out.println("Parent class Constructor");
  }
}
class Subclass extends Parentclass
{
  Subclass()
  {
    System.out.println("Child class Constructor");
  }
  Subclass(int num)
  {
    System.out.println("Parameterized Constructor of Child class");
  }
  void display()
  {
    System.out.println("Hello from display method of Child class");
  }
  public static void main(String args[])
  {
    Subclass obj= new Subclass();
    obj.display();
    Subclass obj2= new Subclass(10);
    obj2.display();
  }
}
```

Output:

```
Parent class Constructor
Child class Constructor
Hello from display method of Child class
```

Example: Invoking the parameterized constructors of the super class

```
class Parentclass
{
```

```
    Parentclass()
    {
       System.out.println("No-argument constructor of parent class");
    }
    Parentclass(String str)
    {
       System.out.println("Parameterized constructor of parent class");
    }
}
class Subclass extends Parentclass
{
  Subclass()
  {
  /* super() must be added to the first statement of constructor otherwise you
      will get a compilation error. When we explicitly use super in constructor
      the compiler doesn't invoke the parent constructor automatically.*/
        super("Hello from subclass constructor");
        System.out.println("Constructor of child class");
  }
  void display()
  {
        System.out.println("Hello from display method of subclass");
  }
  public static void main(String args[])
  {
        Subclass obj= new Subclass();
        obj.display();
  }
}
```

Output:

Parameterized constructor of parent class
Constructor of child class
Hello from display method of subclass

2.2.4 final keyword:

final is a non-access modifier for Java elements. The final modifier is used for finalizing the implementations of classes, methods, and variables.

- A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to a different object.
- A final method cannot be overridden. Though a sub class can call the final method of parent class without any issues but it cannot override it. The main intention of making

a method final would be that the content of the method should not be changed by any outsider.

- The main purpose of using a class being declared as final is to prevent the class from being sub-classed. If a class is marked as final then no class can inherit any feature from the final class.

Points to remember:

- A constructor cannot be declared as final.
- Local final variable must be initializing during declaration.
- All variables declared in an interface are by default final.
- We cannot change the value of a final variable.
- A final method cannot be overridden.
- A final class not be inherited.

Example: final variable

```
public class ABC
{
 final int x = 10;
 public static void main(String[] args)
 {
  ABC obj = new ABC();
  obj.x = 20; // will generate an error: cannot assign a value to a final variable
  System.out.println(obj.x);
 }
}
```

Output:

```
error: cannot assign a value to final variable x
   obj.x = 20; // will generate an error: cannot assign a value to a final variable
     ^
1 error
```

Example: final method

```
class Car
{
 final void run()
 {
  System.out.println("Car");
 }
}
class  Maruti extends Car
{
```

```
  void run()
  {
   System.out.println("Speedy");
  }
  public static void main(String args[])
  {
   Maruti Baleno= new Maruti();
   Baleno.run();
  }
}
```

Output:

```
error: run() in Maruti cannot override run() in Car
  void run()
       ^
  overridden method is final
1 error
```

Example: final class

```
final class FC
{
}
class ABC extends FC
{
  void method()
  {
    System.out.println("My Method");
  }
  public static void main(String args[])
  {
    ABC obj= new ABC();
    obj.method();
  }
}
```

Output:

```
error: cannot inherit from final FC
class ABC extends FC
                 ^
1   error
```

2.2.5 Abstract class:

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Abstraction lets you focus on what the object does instead of how it does it. Abstraction can be achieved with one of two ways: Abstract class and Interface

The abstract keyword is a non-access modifier, used for classes and methods:

Abstract class: A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated. To access the abstract class, it must be inherited from another class.

Key features of an abstract class:

- Abstract classes may or may not contain abstract methods.
- Abstract classes can have constructors and variables, just like other normal classes.
- If a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- A class can inherit only one abstract class, as multiple inheritance is not allowed amongst classes.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstract method: A method without body (no implementation) is known as abstract method. It can only be used in an abstract class, and it does not have a body. The body is provided by the subclass.

Key features of an abstract method:

- Abstract methods don't have body, they just have method signature.
- If a class has an abstract method it should be declared abstract, but the vice versa is not true, which means an abstract class doesn't need to have an abstract method compulsory.
- If a regular class extends an abstract class, then the class must have to implement all the abstract methods of abstract parent class or it has to be declared abstract as well.

Why abstract methods?

If we want to force the same name and signature pattern in all the subclasses and do not want to give them the opportunity to use their own naming conventions, but at the same time give them the flexibility to code these methods with their own specific requirements.

Example:

```
abstract class Animal
{
  public abstract void sound();
}
public class Dog extends Animal
{
  public void sound()
  {
        System.out.println("Dog says: Bow Wow");
  }
  public static void main(String args[])
  {
        Animal obj = new Dog();
        obj.sound();
  }
}
```

Output:

Dog says: Bow Wow

Example:

```
abstract class Car
{
 Car()
 {
  System.out.println("Constructor Car");
 }
 abstract void run();
 void changeGear()
 {
  System.out.println("Gear changed");
 }
}
class Maruti extends Car
{
 void run()
  {
  System.out.println("implementing run method");
  }
}
class AbstractDemo
{
```

```
public static void main(String args[])
{
 Maruti obj = new Maruti();
 obj.run();
 obj.changeGear();
}
}
```

Output:

Constructor Car
implementing run method
Gear changed

2.3 Interface:

Another way to achieve abstraction in Java, is with interfaces. It is used to achieve abstraction and multiple inheritance in Java. interfaces can have abstract methods and variables. It cannot have a method body. It cannot be instantiated just like the abstract class. The interface keyword is used to declare an interface. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements. Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

Interfaces have the following properties:

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Syntax:

```
interface interface_name
{
   // declare constant fields
   // declare abstract methods
}
```

Implementing Interfaces

A class uses the implements keyword to implement an interface. A class that implements an interface must implement all the methods declared in the interface. The methods must have the exact same signature (name + parameters) as declared in the interface. The class does not need to implement (declare) the variables of an interface, but only the methods.

When implementation interfaces, there are several rules:

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend another class.

Syntax:

```
interface interface_name
{
   return_type methodname(argumentlist);
   ..........
}
class class_name implements interface_name
{
}
```

Example:

```
interface Calculate_Area
{
 final float PI=3.14f;
 public float area(int x);
 public void shape();
}
class Square implements Calculate_Area
{
 public float area(int x)
 {
  return x*x;
 }
 public void shape()
 {
  System.out.println("From Square");
 }
}
```

```
class Circle implements Calculate_Area
{
 public float area(int x)
 {
 return PI*x*x;
 }
 public void shape()
 {
 System.out.println("From Circle");
 }
}
class Interface_Demo
{
 public static void main(String args[])
 {
 Square s=new Square();
 System.out.println("Area of Square: " + s.area(2));
 s.shape();
 Circle c=new Circle();
 System.out.println("Area of Circle: " + c.area(2));
 c.shape();
 }
}
```

Output:

```
Area of Square: 4.0
From Square
Area of Circle: 12.56
From Circle
```

2.3.1 Variables in Interface:

Just like methods in an interface (by default public and abstract), variables defined i an interface also carry a default behavior. They are implicitly public, final and stati and there is no need to explicitly declare them.

- As they are final, they need to be assigned a value compulsorily.
- As they are static, they can be accessed directly with the help of an interfac name.
- As they are public, we can access then from anywhere.

Example:

```
interface X
{
```

```
        int x = 20;
}
class VarIn implements X
{
        public void getX()
        {
                System.out.println(x);
        }
}
class Main
{
        public static void main(String args[])
        {
                VarIn obj = new VarIn();
                obj.getX();
                //obj.x++; //uncommenting this will generate compile error
        }
}
```

Output:
20

2.3.2 Extending Interfaces:

An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface. A Java class can only extend one parent class.

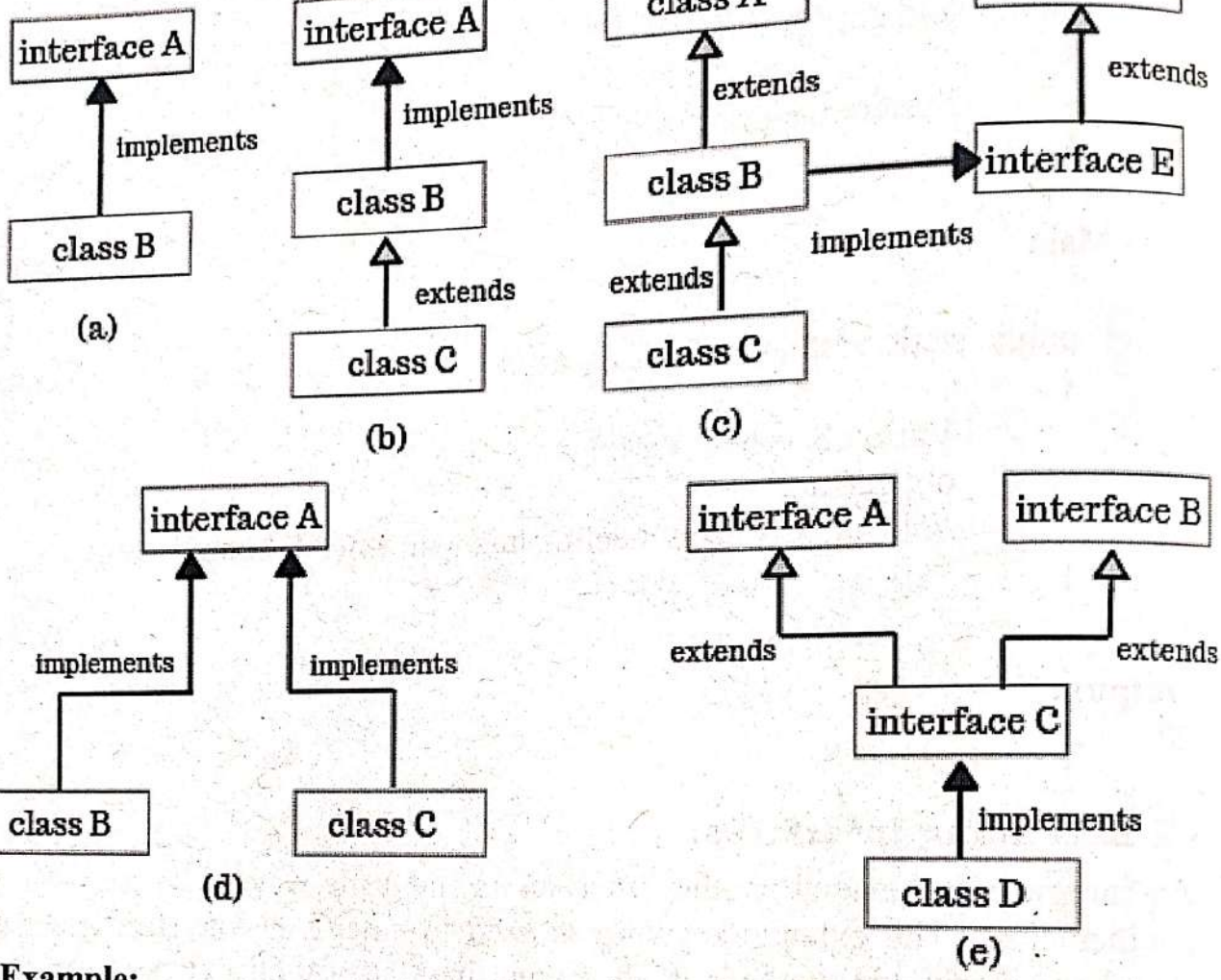
When one interface inherits from another interface, that sub-interface inherits all the methods and constants that its super interface declared. In addition, it can also declare new abstract methods and constants.

Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface. The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

Syntax:

```
[public] interface Interface_Name extends interface1[, interface2, interfaceN]
{
 //interface body
}
```

Various forms of extending interfaces and Relationship between classes and interfaces



Example:

```
interface One
{
 public void msg1();
}
interface Two extends One
{
 public void msg2();
}
public class Interface_Demo1 implements Two
{
 public void msg1()
 {
  System.out.println("Hello from msg1");
 }
 public void msg2()
 {
```

```
    System.out.println("Hello from msg2");
   }
   public static void main(String args[])
   {
   Interface_Demo1 obj=new Interface_Demo1();
   obj.msg1();
   obj.msg2();
  }
}
```

Output:

Hello from msg1
Hello from msg2

Example: implementing multiple inheritance using multiple interfaces

```
interface Home
{
 void homeLoan();
}
interface Car
{
 void carLoan();
}
interface Education
{
 void educationLoan();
}
public class Loan implements Home, Car, Education
{
  public void homeLoan()
  {
    System.out.println("Rate of interest on home loan is 16.50%");
  }
  public void carLoan()
  {
    System.out.println("Rate of interest on car loan is 11.25%");
  }
  public void educationLoan()
  {
    System.out.println("Rate of interest on education loan is 10.50%");
  }
  public static void main(String[] args)
```

```
    {
      Loan l=new Loan();
      l.homeLoan();
      l.carLoan();
      l.educationLoan();
    }
}
```

Output:

Rate of interest on home loan is 16.50%
Rate of interest on car loan is 11.25%
Rate of interest on education loan is 10.50%

Interface vs Abstract class:

Interface	Abstract class
Interface is a Java Object containing method declaration but no implementation. The classes which implement the Interfaces must provide the method definition for all the methods.	Abstract class is a class which contain one or more abstract methods, which has to be implemented by its sub classes.
Interface is a pure abstract class which starts with interface keyword.	Abstract class is a Class prefix with an abstract keyword followed by Class definition.
Interface contains all abstract methods and final variable declarations.	Abstract class can also contain concrete methods.
Methods in an interface cannot be static.	Non-abstract methods can be static.
All methods of an interface need to be overridden.	Only abstract methods need to be overridden.
Interfaces are useful in a situation that all properties should be implemented.	Abstract classes are useful in a situation that Some general methods should be implemented and specialization behavior should be implemented by child classes.
Interface supports multiple inheritance.	Abstract class doesn't support multiple inheritance.
Interface has only static and final variables.	Abstract class can have final, non-final, static and non-static variables.
Interface can't provide the implementation of abstract class.	Abstract class can provide the implementation of interface.

Members of a Java interface are public by default.	A Java abstract class can have class members like private, protected, etc.

Example:
public interface ABC
{
void method();
}
	Example:
	public abstract class PQR
	{
	public abstract void method();
	}

Exercises

Answer the following question.

1. Define array. How to declare array in JAVA?
2. Explain passing arrays to methods and returning arrays from method with an example.
3. Discuss for-each/enhanced loop with arrays.
4. What are command-line arguments and how are they used?
5. Write a note on inheritance. Explain method overriding with a suitable example.
6. Explain the use of super keyword with an example.
7. Explain how to restrict method overriding in a subclass.
8. Write a note on abstract class with a suitable example.
9. Differentiate between abstract class and interface.
10. How to achieve multiple inheritance in JAVA? – Explain with a suitable example.

Fill in the Blanks.

1. In JAVA, arrays are _____. **(objects)**
2. char[] c=new char[5]; is a valid statement. **(true)**
3. What is the output of following code? **(0)**

```
public class Test
{ public static void main(String[] args)
    { int[] a = new int[0];
      System.out.print(a.length); }
    }
```

4. Arrays in JAVA are dynamically allocated using the _____ operator. **(new)**

5. Because Java does not support pointers, Java array elements are accessed only through indexes. **(true)**
6. It is necessary to implement all methods in an interface. **(yes)**
7. _____ is the default access modifier for an interface method. **(public)**
8. How many concrete classes can you have inside an interface? **(none)**
9. If you do not implement all the methods of an interface while implementing, what specifier should you use for the class? **(abstract)**
10. We can define a variable in an interface of type _____ and _____. **(final and static)**
11. We can achieve multiple inheritance in Java using _____. **(interfaces)**
12. Interfaces can't be extended. **(false)**
13. _____ keyword is used to inherit a class. **(extends)**
14. Final methods can be overridden. **(false)**
15. Private members of a class are inherited to sub class. **(false)**
16. A class can be declared as _____ if you do not want the class to be sub-classed. **(final)**
17. If a class that implements an interface does not implement all the methods of the interface, then the class becomes a/an _____ class. **(abstract)**
18. Inheritance relationship in Java language is _____. **(is-a)**
19. Which of the keyword is used to define interfaces in Java? **(interface)**
20. Which of the keyword is used by a class to use an interface defined previously? **(implements)**
21. Can we write explicit constructors in an abstract class? **(yes)**
22. Can we declare protected methods in an interface? **(no)**
23. Can we declare abstract methods as static? **(no)**
24. Abstract methods can be declared as final. **(final)**
25. An abstract class contains at least _____ abstract methods. **(one)**
26. The body of an abstract method is defined in the _____. **(subclass)**
27. An interface contains _____ abstract methods. **(all)**
28. An interface does not contain any _____ variable; it contains only constant variables. **(instant)**
29. All the methods of an interface are automatically _____. **(public)**
30. The keyword _____ always refers to the superclass of the class in which you use it. **(super)**

CC-214 Core Java Practical (UNIT – 2)

1. Write a program to sort the elements of one dimensional array. Read value of array elements through command line argument.

```
public class SortArray
{
        public static void main(String[] args)
        {
                int i, j, n, temp;
                n=args.length;
                int a[] = new int[n];
                for(i=0; i<n; i++)
                {
                  a[i]=Integer.parseInt(args[i]);
                }
                for (i=0; i<n; i++)
                {
                        for (j=i+1; j<n; j++)
                        {
                                if (a[i] > a[j])
                                {
                                        temp = a[i];
                                        a[i] = a[j];
                                        a[j] = temp;
                                }
                        }
                }
                System.out.println("Array Elements after sorting:");
                for (i=0; i<n; i++)
                {
                        System.out.println(a[i]);
                }
        }
}
```

Output:

```
D:\javapr>javac SortArray.java
D:\javapr>java SortArray 6 4 0 -1 2
Array Elements after sorting:
```

-1
0
2
4
6

2. Write a program to create an array to store 5 integer values. Also initialize the array with 5 numbers and display the array Elements in reverse order.

```
public class ReverseArray
{
  public static void main(String[] args)
  {
    int [] arr = new int [] {1, 2, 3, 4, 5};
    System.out.println("Original array: ");
    for (int i = 0; i < arr.length; i++)
    {
      System.out.print(arr[i] + " ");
    }
    System.out.println();
    System.out.println("Array in reverse order: ");
    for (int i = arr.length-1; i >= 0; i--)
    {
      System.out.print(arr[i] + " ");
    }
  }
}
```

Output:

Original array:
1 2 3 4 5
Array in reverse order:
5 4 3 2 1

3. Write a program to find sum of two matrices of 3 x 3.

```
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]+"\t");
  }
    System.out.println();
  }
 }
}
```

Output :

```
Enter the number of rows and columns of matrix:
3 3
Enter the elements of first matrix:
1 2 3
4 5 6
1 1 1
Enter the elements of second matrix:
1 1 1
1 2 3
4 5 6
Sum of the matrices:
2    3    4
5    7    9
5    6    7
```

4. Write program to create an array of company name and another array of price quoted by the company. Fetch the company name who has quoted the lowest amount.

```
class Quote
{
  public static void main (String[] args)
  {
    int N,I;
    String Company[]={"LG","SAMSUNG","SONY","PHILIPS","MI"};
    int price[]={1000,2000,1500,500,200};
    N=price.length;
    int low=price[0];
    for(I=1;I<=N-1;I++)
    if(low>price[I])
    low=price[I];
    for(I=0;I<=N-1;I++)
    if(low==price[I])
    System.out.println(Company[I]);
  }
}
```

Output:

```
MI
```

5. Write an interface called numbers, with a method in Process(int x, int y). Write a class called Sum, in which the method Process

finds the sum of two numbers and returns an int value. Write another class called Average, in which the Process method finds the average of the two numbers and returns an int.

```
interface Numbers
{
public int process(int x,int y);
}
class Sum implements Numbers
{
public int process(int x,int y)
{
return(x+y);
}
}
class Average implements Numbers
{
public int process(int x,int y)
{
return((x+y)/2);
}
}
class InterfaceDemo
{
public static void main(String args[])
{
int a,b;
Sum add=new Sum();
a=add.process(10,20);
System.out.println("Your Sum is:"+a);
Average avg=new Average();
b=avg.process(40,80);
System.out.println("Your Average is:"+b);
}
}
```

Output:

Your Sum is:30
Your Average is:60

6. **Declare an abstract class Vehicle with an abstract method named numWheels(). Provide subclasses Car and Truck that each**

implements this method. Create instance of these subclasses and demonstrate the use of this method.

```
abstract class vehicle
{
  public abstract void numwheels();
}
class Car extends vehicle
{
  public void numwheels()
  {
    System.out.println("Car has four wheels.");
  }
}
class Truck extends vehicle
{
  public void numwheels()
  {
    System.out.println("Truck has six wheels.");
  }
}
public class Demo
{
  public static void main( String args[])
  {
    Car c= new Car();
    c.numwheels();
    Truck t= new Truck();
    t.numwheels();
  }
}
```

Output:

Car has four wheels.
Truck has six wheels.

7. Write an interface called Exam with a method Pass(int mark) tha returns a Boolean. Write another interface called Classify with method Division(int average) which returns a string. Write a clas called Result which implements both Exam and Classify. The pas method should return true if the marks is greater than or equa to 35 else false. The division method must return "First" whe the parameter average is 60 or more, "second" when average i

50 or more but below 60, "no division" when average is less than 50.

```
import java.io.*;
interface Exam
{
        boolean Pass(int mark);
}
interface Classify
{
        String Division(int avg);
}
class Result implements Exam,Classify
{
        public boolean Pass(int mark)
        {
                if(mark>=50)
                        return true;
                else
                        return false;
        }
        public String Division(int avg)
        {
                if(avg>=60)
                        return "First";
                else if(avg>=50)
                        return "Second";
                else
                        return "No-Division";
        }
}
public class MyResult
{
        public static void main(String[] args)
        {
                boolean pass;
                int mark,avg;
                String division;
                DataInputStream in=new DataInputStream(System.in);
                Result res=new Result();
                try
```

```
            {
                System.out.println("Enter the mark : ");
                mark=Integer.parseInt(in.readLine());
                System.out.println("Enter the average : ");
                avg=Integer.parseInt(in.readLine());
                pass=res.Pass(mark);
                division=res.Division(avg);
                if(pass)
                    System.out.println("Passed - "+ division + ".");
                else
                    System.out.println("Failed - " + division+ ".");
            }
            catch(Exception e)
            {
                System.out.println("Error : " + e);
            }
        }
}
```

Output:

```
Enter the mark :
80
Enter the average :
70
Passed - First.
```

8. Create class calculation with an abstract method area( ). Creat Rectangle and Triangle subclasses of calculation and find area rectangle and triangle.

```
abstract class Shape
{
 abstract void area();
 double area;
}
class Triangle extends Shape
{
 double b=60,h=20;
 void area()
 {
  area = (b*h)/2;
  System.out.println("Area of Triangle = "+area);
```

```
 }
}
class Rectangle extends Shape
{
 double w=80,h=10;
 void area()
 {
  area = w*h;
  System.out.println("Area of Rectangle = "+area);
 }
}
class Area
{
 public static void main(String [] args)
 {
  Triangle t= new Triangle();
  Rectangle r =new Rectangle();
  t.area();
  r.area();
 }
}
```

Output:

Area of Triangle = 600.0
Area of Rectangle = 800.0

9. The abstract Vegetable class has four subclasses named cabbage, carrot and potato. Write an application that demonstrates how to establish this class hierarchy. Declare one instance variable of type string that indicates the color of a vegetable. Create and display instances of these object. Override the toString() method of object to return a string with the name of the vegetable and its color.

```
abstract class Vegetable
{
public String color;
}
class Potato extends Vegetable
{
 public String toString()
 {
```

```
  color = "Brown -skinned Color";
  return "potato -->"+color;
 }
}
class Brinjal extends Vegetable
{
 public String toString()
 {
  color = "purple color";
  return "Brinjal -->"+this.color;
 }
}
class Tomato extends Vegetable
{
 public String toString()
 {
  color="red color";
  return "Tomato -->"+color;
 }
}
class VegCol
{
 public static void main(String [] args)
 {
  Potato p = new Potato();
  Brinjal b = new Brinjal();
  Tomato t = new Tomato();
  System.out.println(p);
  System.out.println(b);
  System.out.println(t);
 }
}
```

Output:

```
potato -->Brown -skinned Color
Brinjal -->purple color
Tomato -->red color
```

Unit – 3

Packages, String and Exception Handling

- Package
- Java.lang. Package
- Java.lang.Objects Class
- Java.wrapper classes
- String class
- String Buffer class
- Exception

Unit-3 Package, String and Exception Handling

3.1 Package:

Java is object oriented programming language. The feature of object oriented programming is to provide reusability. The code reusability is possible with inheritance. In java code reusability is possible with package.

The package is collection or related classes and interfaces. If we want to use a class then we can import the package and can use the class from that package.

3.1.1 Types of Packages:

In java there are different types of packages.

1. Inbuilt package.
2. User defined package.

Inbuilt packages are provided by java itself. Java provides many packages to perform different kind of tasks. For example to write program of applet java provides .applet package, to perform networking task, java provides .net package.

Programmer can create their own packages in java that provides reusability and that is called user defined package.

3.1.2 Creating Packages:

To create user defined package different steps to be followed.

3.1.2.1 Package statement:

To create user defined package in java, package keyword is used. This is the first statement of any class that we want to create inside the package.

It is possible in java to create multiple source files, that is java files and we can put it together in same package. To put it in the same package, they must have first statement is package and the name of package in the java file.

For example, suppose we want to create package p1 that contains different classes like X,Y and Z then we create different files like, X.java that contains source code for the class X, same for Y and Z. in all the source files the first statement should be,

```
package p1;
```

3.1.2.2 Store the classes in the package:

A single package may contains many classes and interfaces. In above example class X, Y and Z can be created in different source files. This files are part of package p1, so this files should be saved in single directory called p1. The classes are in different source files so, the all classes can be public. It is possible to store more than one

classes are stored in single source file, in such case only one class can be public at a time. For example suppose we have created two java files called X.java and YZ.java where, X.java file contains class X and YZ.java file contains Y and Z classes then, either of the classes can be public. If we want to make both classes public they must be saved in different source files.

3.1.2.3 Compilation of classes:

To compile the classes from the package, open the command prompt and navigate up to the directory and use command to compile. For example suppose we have created package p1 with classes X, Y and Z and if we want to compile the class X of package p1 then, we should use command as mentioned in following figure.

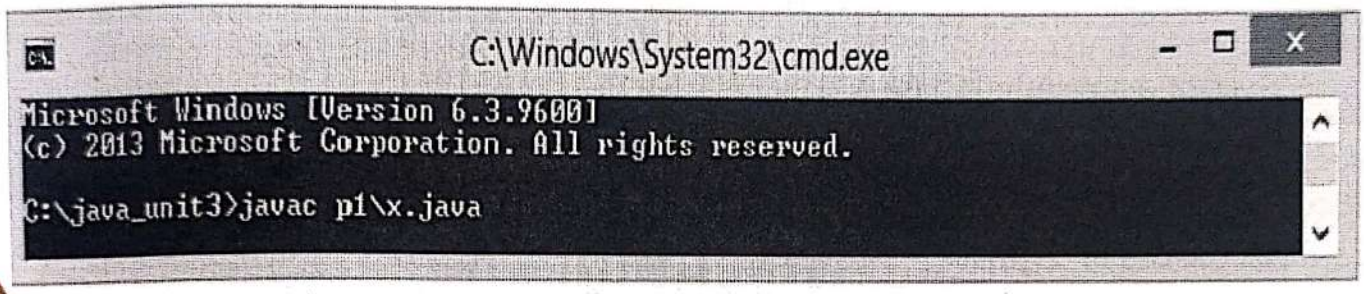


Figure 3.1

If we want to compile all classes of the package together then we can write following command.

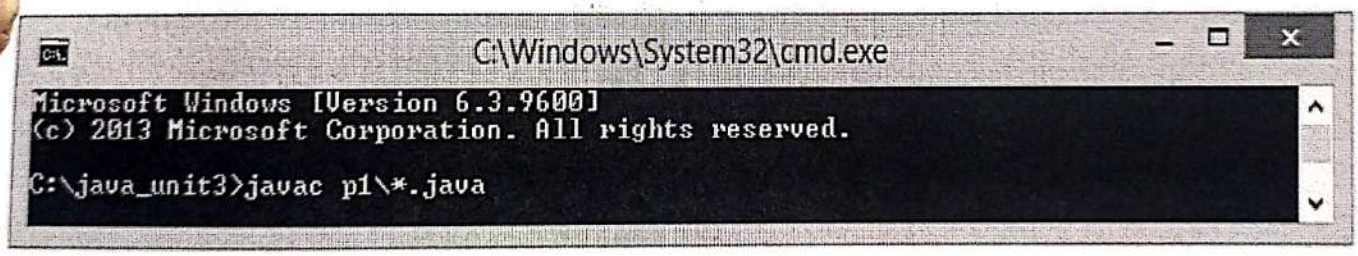


Figure 3.2

3.1.2.4 Execution of the classes:

The class that belongs to package can be executed as mentioned in below figure.

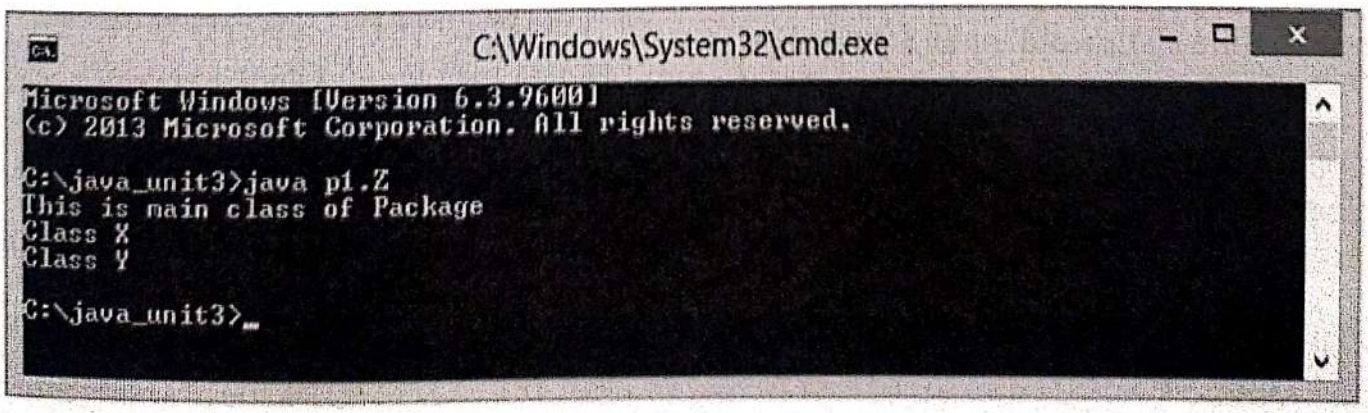


Figure 3.3

Here , we have mentioned the steps to create a package. Now let we go through the complete source code of the package program. As per our example here we have created three different source files; X.java, Y.java and Z.java.

```
//Program of package
//Source file of X.java
package p1;
class X
{
  void x1()
  {
   System.out.println("Class X");
  }
}
//Source file of Y.java
package p1;
class Y
{
  void y1()
  {
   System.out.println("Class Y");
  }
}
//Source file of Z.java
package p1;
class Z
{
  public static void main(String args[])
  {
   System.out.println("This is main class of Package");
   X  obj1= new X();
   obj1.x1();
   Y obj2 = new Y();
   obj2.y1();
  }
}
```

In above code class X contains x1 method and prints class X, class Y contains y1 method and prints class Y and class Z creates instances of class X and Y both and call the respective methods of both classes.

The output of above code is :

This is main class of Package

Class X

Class y

3.1.3 Using Packages:

In java with use of package we can easily access any classes or interfaces. To use any package import statement is used.

In above program the class X, Y and Z belongs to same package so, import statement is not required and class directly objects of X and Y can be created in class Z. However, if the classes belongs to different packages then import statement is required.

Let we take an example for java inbuilt packages, suppose we want to use the class from inbuilt package applet then we can write,

Import java.applet .*;

Through above statement the class Applet form applet package can be used easily. If multiple classes are required from different packages then, multiple import statement can be written in single java file.

For example,

Import java.applet.*;

Import java.awt.*;

In the above code all the classes, interfaces and sub pakcages of awt package are also part of our program. If we required particular class only of the package then that is also possible.

Import java.awt.Graphics;

In above code only Graphics class is imported from awt package.

When we import the package the public classes are accessible to another package. The non public classes can not be accessed in another package the classes are hidden from being accessed by the importing class.

Here is an example:

package p1;

```
public class x
{
...
}
class y
{
...
}
class z
{
...
}
```

Now suppose we import package p1 in Mypack.java as describe below:

```
import p1.*;
class Mypack
{
  Public static void main(String args[])
 {
  ...
}
}
```

In Mypack class, we can not access the Y and Z classes as they are not public i package p1 and these classes can be described as hidden classes. Following topi describes in depth, how can we access different classes from the package.

3.1.4 Access Protection:

The scope of variable, method or class can be defined with use of access modifiers. I java, there are four types of Access modifiers:

1. Private
2. Protected
3. Public
4. Default

When private access specification has been used means it can be accessed within th same class only. Further access is not allowed for the private variables or methods Private data can not be accessed in classes of same package or classes of othe package.

The protected access specification can be used with classes, methods or variables. Protected can be accessed in the same class, it can be accessed in all classes in the same package and sub class of other package.

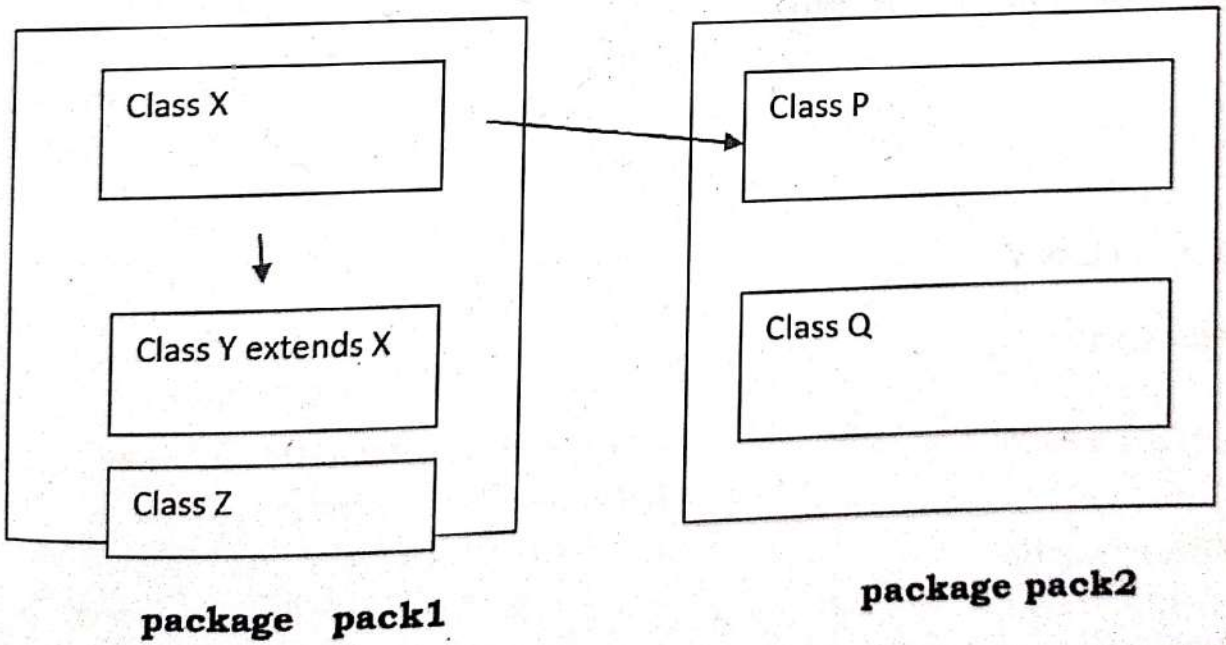
Public can be accessed anywhere. In the same class, same package or in other package even.

Default access specification can be accessed in all classes of the same package but cannot be accessed in other package.

Following table describes access specification.

Access specifier	In same class	None Sub class in same package	Sub class in same package	None sub class in other package	Sub class in other package
Private	Accessible	Not Accessible	Not Accessible	Not Accessible	Not Accessible
Default	Accessible	Accessible	Accessible	Not accessible	Not accessible
Protected	Accessible	Accessible	Accessible	Not accessible	Accessible
public	Accessible	Accessible	Accessible	Accessible	Accessible

Now let we learn access specification with an example:



Following program describes access specification. Here two packages are created; pack1 and pack2.

In package, pack1 X, Y and Z classes are created and in package, pack2 class P and class Q are created. Class Y is inherited from class x and class P is also inherited from class X.

Class X has private, protected, public and default member data and the access level is described in different classes in following program.

// Code of class X

```
package pack1;
public class X
{
   private int a=5;
   protected int b=10;
   public int c=15;
   int d=20;
  public void x1()
  {
   System.out.println("Class X");
   System.out.println(" a is" +a);
   System.out.println(" b is" +b);
   System.out.println(" c is" +c);
   System.out.println(" d is" +d);

  }
}
```

// Code of class Y

```
package pack1;

class Y extends X
{
  public void y1()
  {
    System.out.println("Class Y");
//  System.out.println(" a is" +a); Not accessible as it is private
   System.out.println(" b is" +b);
```

```
    System.out.println(" c is" +c);
    System.out.println(" d is" +d);
  }
}
```

//Code of class Z

```
package pack1;

class Z
{
  public void z1()
 {
    X obj1= new X();
   System.out.println("Class Z");
  // System.out.println(" a is" +obj1.a); Not accessible as it is private
   System.out.println(" b is" +obj1.b);
   System.out.println(" c is" +obj1.c);
   System.out.println(" d is" +obj1.d);
 }
}
```

// code of class P

```
package pack2;
import pack1.*;

class P extends X
{
  void p1()
 {
   System.out.println("Class P");
//  System.out.println(" a is" +a); Not accessible
  System.out.println(" b is" +b);
  System.out.println(" c is" +c);
//  System.out.println(" d is" +d);
 }
}
```

```
//Code of class Q
package pack2;
import pack1.*;
class Q
{
  void q1()
  {
    X obj1= new X();
   System.out.println("Class  Q");
  // System.out.println(" a is" +obj1.a); Not accessible private data
  // System.out.println(" b is" +obj1.b); Not accessible protected data
   System.out.println(" c is" +obj1.c);
   //System.out.println(" d is" +obj1.d); Not accessible default data
  }
}
```

```
//code of Pack2_main class
package pack2;
import pack1.*;
class Pack2_main
{
  public static void main(String args[])
  {
    X obj1= new X();
    Y obj2= new Y();
    Z obj3= new Z();
    P obj4 = new P();
    Q obj5=new Q();
    System.out.println("Class Pack2_main");

    obj1.x1();
    obj2.y1();
    obj3.z1();
    obj4.p1();
    obj5.q1();
   }
}
```

Output:

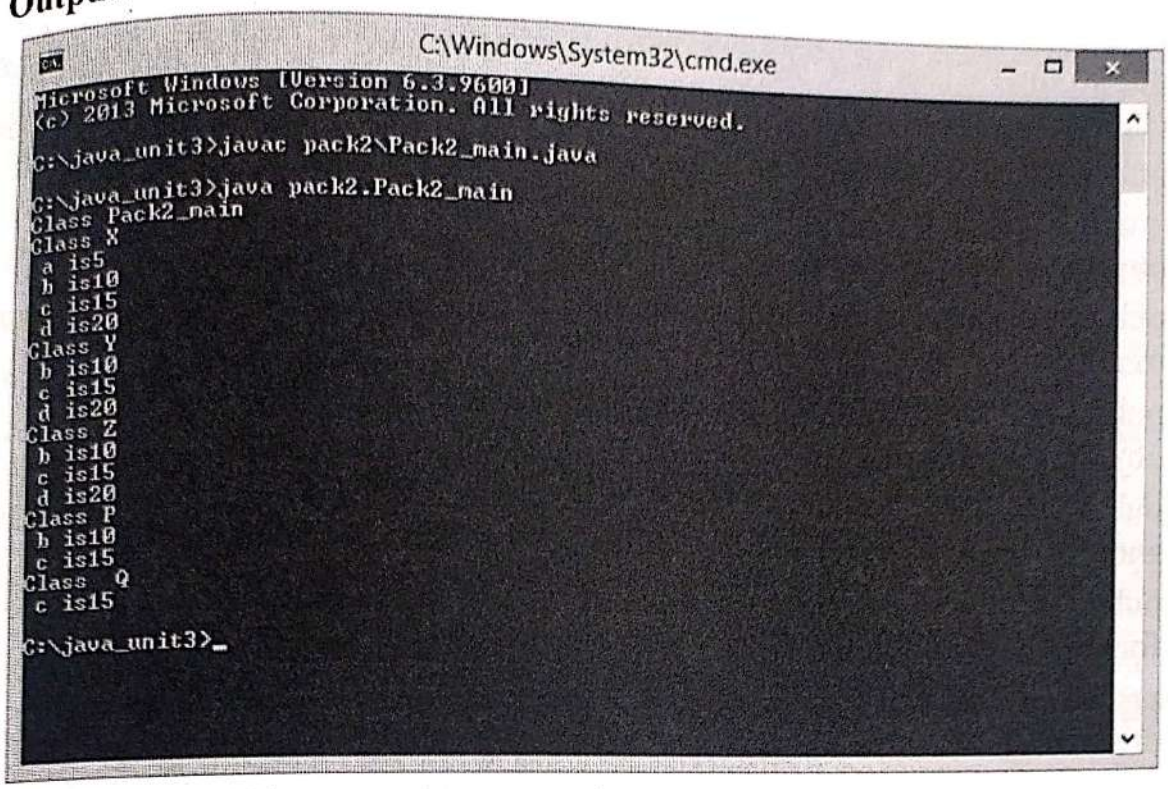


Figure 3.4

3.2 Java.lang. Package:

lang package is default package that is part of every program though it is not imported in the program. Lang package provides fundamental classes for java language.

The lang package contains many classes like, Boolean, Byte, Character, Compiler, Double , Integer, Math,Long , Number, Object , Thread, String and many more classes are part of it.

The Byte, Integer, Boolean, Long, Double , Character classes are wrapper classes for primitive data types byte, int, boolean, long, double and character.

The object class from lang package is ultimate base class of all java classes.

The Thread class provides support for multi threading programming.

The Math class provides the mathematical methods.

To support string, String class is available and classes are available that are used to access system resources.

3.3 Java.lang.Objects Class:

The Object class is very important class of lang package. The Object class is root class of class hierarchy. Object class is used to represent primitive type. The object class can be used to identify the object type.

The object class provides many methods. Some methods described below:

1. public final Class getClass() : This method is used to find the details about the class.
2. public booelan equals(Object obj): This method is used to compare two objects.
3. public String toString(): This method returns the string representation of the object.
4. public final void notify(): this method wakes up single thread, waiting on object's monitor.
5. public final void notifyAll(): This method wakes up all thread, waiting on this object's monitor.
6. public final void wait(): This method causes the current thread to wait, until another thread notifies.
7. public final void wait(long timeout): This method causes the current thread to wait for specified milliseconds.
8. protected Object clone() throws CloneNotSupportedException: This method create and return the clone of the current object.
9. protected void finalize() throws Throwable: This method is invoked by garbage collector before object is being garbage collected.
10. public int hashCode(): This method returns the hashcode number for the current object.

3.4 Java.wrapper classes:

Java wrapper class provides facility to use primitive data types. Wrapper class provide object representation for byte, short,int , long,float,double, character and boolean data types. Wrapper classes are immutable and final. These classes provides numerous methods with use of that we can work with primitive types very easily.

Wrapper class support autoboxing and unboxing. Autoboxing is conversion of primitive data type into an object of the corresponding wrapper class. Unboxing is converting an object of wrapper type to its corresponding object type. Number class is base class of all Wrapper classes. This is an abstract class. Following table describes the primitive data types and its relevant wrapper classes.

Data type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
character	Character

Let we learn about all wrapper classes in depth.

3.4.1 Byte class:

Byte class supports byte primitive data type. This class provides many methods to support conversion of byte to String, String to byte and byte to other primitive data types. With use of this class byte data type can be converted to Byte object. The constructors, methods and constants of Byte class are described below:

Constructors of Byte class:

Byte(byte bytevalue)

This constructor takes byte primitive data type as an argument and generates Byte class type object.

Byte(String stringvalue)

This constructor construct Byte type object from String type arguments.

Methods of Byte class:

Method prototype	Method description
byte byteValue()	This method converts Byte type object to byte type value.
static int compare(byte a, byte b)	It compares two byte values numerically.
int compateTo(Byte byteobj)	This method compares two Byte objects and return the difference.
double doubleValue()	This method returns value of this Byte as double.
float floatValue()	This method returns value of this Byte as float.
int intValue()	This method returns value of this Byte as int.
long longValue()	This method returns value of this Byte as long.
static byte parseByte(String s)	This method takes String as an argument and convert it to byte type value.
boolean equals(Object obj)	This method compares two Byte objects and returns true if both are same otherwise it returns false.
static Byte valueOf(byte b)	This method returns Byte object from the specified byte value.
static Byte valueOf(String s)	This method returns Byte object from the specified String value.

Constants:

MIN_VALUE
Public static final byte MIN_VALUE
This is constant minimum value a byte can store. -2^7

MAX_VALUE:
Public static final byte MAX_VALUE
This is constant maximum value a byte can store. $2^7 - 1$

Program:

```
class ByteDemo
{
public static void main(String args[])
 {
 byte b11=123;
 Byte o1=new Byte(b11);
 Byte o2=new Byte("123");
 System.out.println("maximum value"+Byte.MAX_VALUE);
 System.out.println("minimum value"+Byte.MIN_VALUE);
 System.out.println("byte value"+o1.byteValue());
 System.out.println("long value"+o2.longValue());
 System.out.println("double value"+o2.doubleValue());
 System.out.println("comparision"+o1.compareTo(o2));
 System.out.println("equals method "+o1.equals(o2));
  byte b4=Byte.parseByte("10");
  System.out.println("b4"+b4);
}
}
```

Output:

3.4.2 Short class:

Short class supports short primitive data type. This class provides many methods to support conversion of short to String, String to short and short to other primitive data types. With use of this class short data type can be converted to Short object. The constructors, methods and constants of Short class are described below:

Constructors of Short class:

Short(short shortvalue)

This constructor takes short primitive data type as an argument and generates Short class type object.

Short(String stringvalue)

This constructor construct Short type object from String type arguments.

Methods of Short class:

Method prototype	Method description
short shortValue()	This method converts Short type object to short type value.
static int compare(short a, short b)	It compares two short values numerically.
int compateTo(Short shortobj)	This method compares two Short objects and return the difference.
double doubleValue()	This method returns value of this Short as double.
float floatValue()	This method returns value of this Short as float.
int intValue()	This method returns value of this Short as int.
long longValue()	This method returns value of this Short as long.
static short parseShort(String s)	This method takes String as an argument and convert it to short type value.
boolean equals(Object obj)	This method compares two Short objects and returns true if both are same otherwise it returns false.
static Short valueOf(short s)	This method returns Short object from the specified short value.
static Short valueOf(String s)	This method returns Short object from the specified String value.

Constants:

MIN_VALUE

Public static final byte MIN_VALUE

This is constant minimum value a short can store. -2^{15}

MAX_VALUE

Public static final byte MAX_VALUE

This is constant maximum value a short can store. $2^{15}-1$

Program:

```
class Shortdemo
{
public static void main(String args[])
{
 Short s1=new Short((short)1234);
 Short s2=new Short("345");

 System.out.println("s1 object"+s1);
 System.out.println("2*s2="+2*s2.shortValue());
 System.out.println("equal"+s1.equals(s2));
 System.out.println("compare"+s1.compareTo(s2));
 System.out.println("string"+s2.intValue());
 }
}
```

Output:

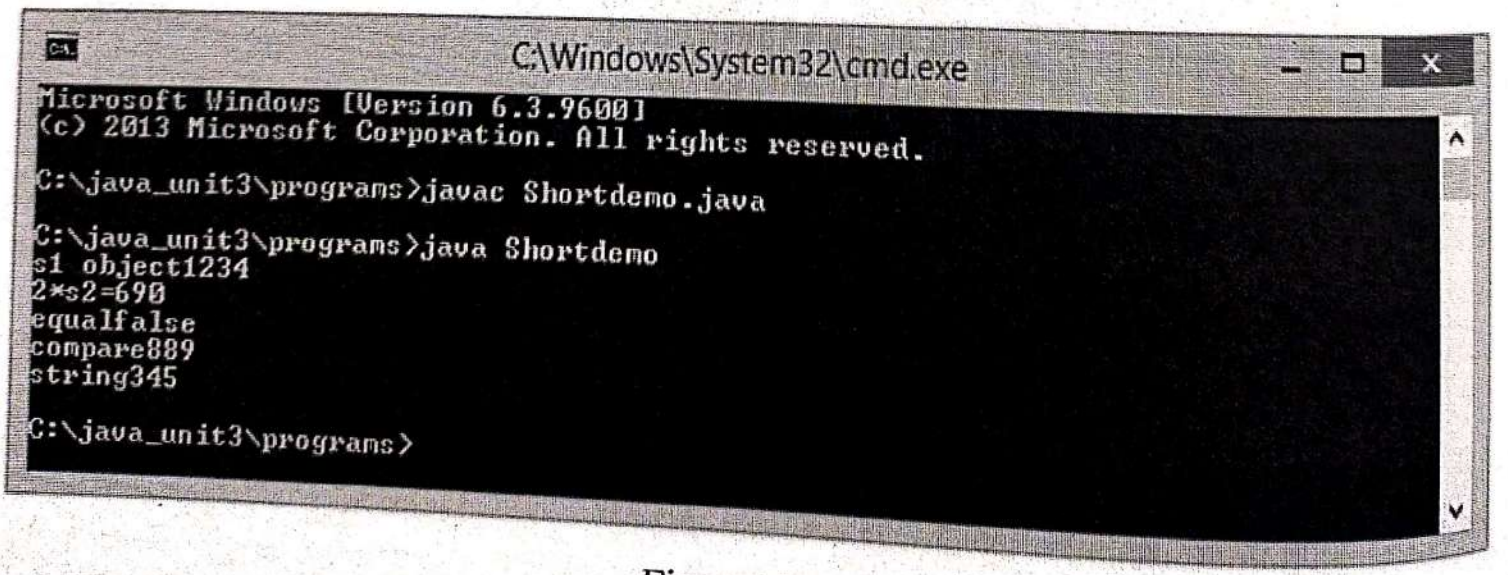


Figure 3.6

3.4.3 Integer class:

Integer class supports int primitive data type. This class provides many methods to support conversion of int to String, String to int and int to other primitive data types. With use of this class int data type can be converted to Integer object. The constructors, methods and constants of Integer class are described below:

Constructors of Integer class:

Integer(int intvalue)

This constructor takes int primitive data type as an argument and generates Integer class type object.

Integer(String stringvalue)

This constructor construct Integer type object from String type arguments.

Methods of Integer class:

Method prototype	Method description
int intValue()	This method converts int type object to int type value.
static int compare(int a, int b)	It compares two int values numerically.
int compateTo(Integer obj)	This method compares two Integer objects and return the difference.
double doubleValue()	This method returns value of this Integer as double.
float floatValue()	This method returns value of this Integer as float.
long longValue()	This method returns value of this Integer as long.
static int parseInt(String s)	This method takes String as an argument and convert it to int type value.
boolean equals(Object obj)	This method compares two Integer objects and returns true if both are same otherwise it returns false.
static Integer valueOf(int s)	This method returns Integer object from the specified int value.
static Integer valueOf(String s)	This method returns Integer object from the specified String value.
static String toBinaryString(int i)	This method returns string in form of unsigned integer in base 2 from int argument.
static String toOctalString(int i)	This method returns string in form of unsigned integer in base 8 from int argument.
static String toHexString(int i)	This method returns string in form of unsigned integer in base 16 from int argument.

Constants:

MIN_VALUE

Public static final int MIN_VALUE
This is constant minimum value a int can store. -2^{31}

MAX_VALUE

Public static final int MAX_VALUE
This is constant maximum value a int can store. $2^{31}-1$

Program:

```
class Intdemo
{
public static void main(String args[])
{
  Integer i1= new Integer(111);
  Integer i2=new Integer("123");

  System.out.println("obj i1"+i1);
  System.out.println("2* i1"+2*i1.intValue());
  System.out.println("equal"+i1.equals(i2));
  System.out.println("compare"+i1.compareTo(i2));
  System.out.println("byte"+i2.byteValue());
  int x=10;
  System.out.println("Integer to binary string "+Integer.toBinaryString(x));
 }
}
```

Output:

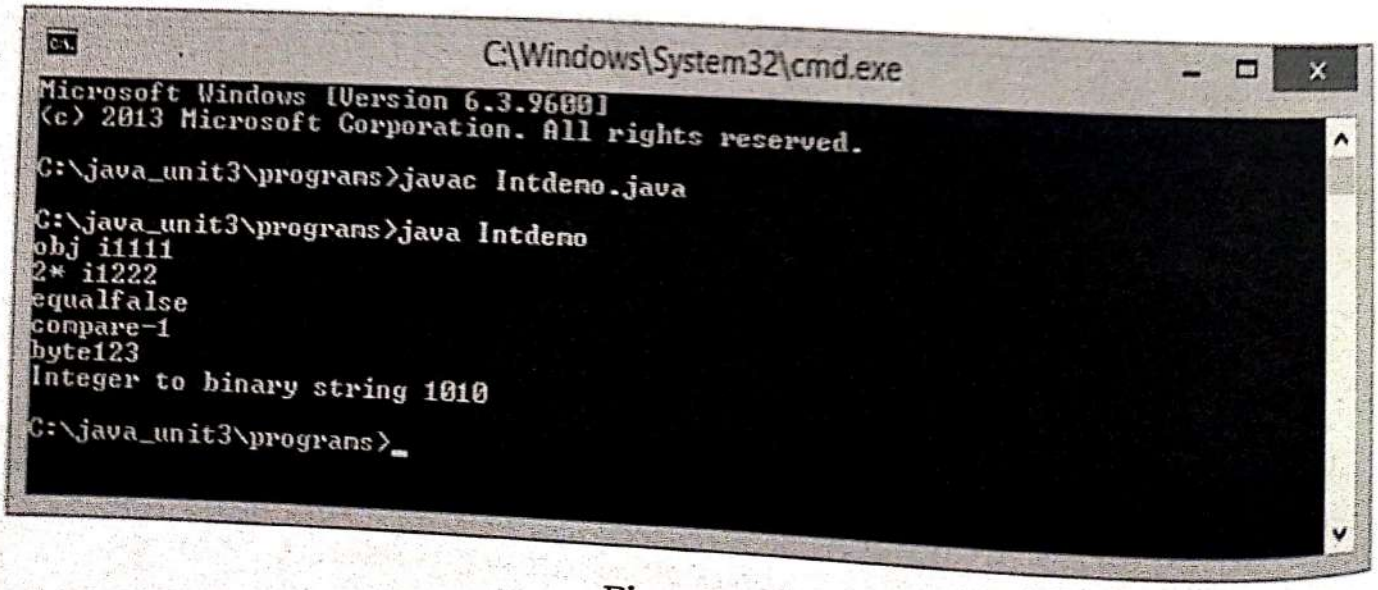


Figure 3.7

3.4.4 Long class:

Long class supports long primitive data type. This class provides many methods to support conversion of long to String, String to long and long to other primitive data types. With use of this class long data type can be converted to Long object. The constructors, methods and constants of Long class are described below:

Constructors of Long class:

Long(long longvalue)

This constructor takes long primitive data type as an argument and generates Long class type object.

Long(String stringvalue)

This constructor construct Long type object from String type arguments.

Methods of Long class:

Method prototype	Method description
long longValue()	This method converts Long type object to long type value.
static int compare(long a, long b)	It compares two long values numerically.
int compateTo(Long obj)	This method compares two Long objects and return the difference.
double doubleValue()	This method returns value of this Long as double.
float floatValue()	This method returns value of this Long as float.
int intValue()	This method returns value of this Long as int.
static long parseLong(String s)	This method takes String as an argument and convert it to long type value.
boolean equals(Object obj)	This method compares two Long objects and returns true if both are same otherwise it returns false.
static Long valueOf(long s)	This method returns Long object from the specified long value.
static Long valueOf(String s)	This method returns Long object from the specified String value.
static String toBinaryString(long i)	This method returns string in form of unsigned integer in base 2 from long argument.
static String toOctalString(long i)	This method returns string in form of unsigned integer in base 8 from long argument.
static String toHexString(long i)	This method returns string in form of unsigned integer in base 16 from long argument.

Constants:

MIN_VALUE

Public static final long MIN_VALUE
This is constant minimum value a long can store. -2^{63}

MAX_VALUE

Public static final long MAX_VALUE
This is constant maximum value a long can store. $2^{63}-1$

Program:

```
class longdemo
{
 public static void main(String args[])
 {
   Long l1= new Long(1234);
   Long l2= new Long("4567");

    System.out.println("obj l1"+l1);
   System.out.println("Object L1, long value " +l1.longValue());
   System.out.println("equal method"+l1.equals(l2));
   System.out.println("compare To method "+l1.compareTo(l2));
   }
}
```

Output:

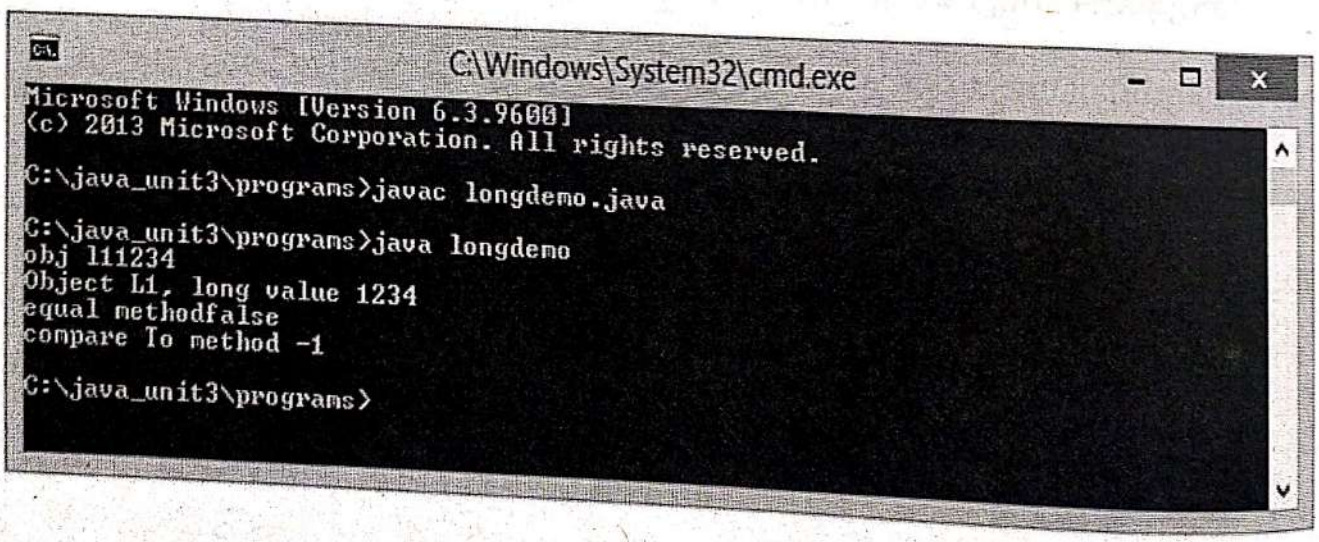


Figure 3.8

3.4.5 Float class:

Float class supports float primitive data type. This class provides many methods to support conversion of float to String, String to float and float to other primitive data types. With use of this class float data type can be converted to Float object. The constructors, methods and constants of Float class are described below:

Constructors of Float class:

Float(float floatvalue)
This constructor takes float primitive data type as an argument and generates Float class type object.

Float(String stringvalue)
This constructor construct Float type object from String type arguments.

Float(double doublevalue)
This constructor convert argument to float type and generates Float class type object.

Methods of Float class:

Method prototype	Method description
float floatValue()	This method converts Float type object to float type value.
static int compare(float a, float b)	It compares two float values numerically.
int compateTo(Float obj)	This method compares two Float objects and return the difference.
double doubleValue()	This method returns value of this Float as double.
static float parseFloat(String s)	This method takes String as an argument and convert it to float type value.
boolean equals(Object obj)	This method compares two Float objects and returns true if both are same otherwise it returns false.
static Float valueOf(float f)	This method returns Float object from the specified float value.
static Float valueOf(String s)	This method returns Float object from the specified String value.

Constants:

MIN_VALUE

Public static final float MIN_VALUE
This is constant minimum value a float can store. 2^{-149}

MAX_VALUE

Public static final float MAX_VALUE
This is constant maximum value a float can store. $(2-2^{-23})\cdot 2^{127}$

Program:

```
class floatdemo
{
public static void main(String args[])
{
  Float f1= new Float(12.34);
  Float f2= new Float("45.67");

  System .out.println("Maximum value "+Float.MAX_VALUE);
  System.out.println("Minimum value " +Float.MIN_VALUE);

  System.out.println("obj f1  "+ f1);
  System.out.println("Object f1, float value  " +f1.floatValue());
  System.out.println("equal method "+f1.equals(f2));
  System.out.println("compare To method "+f1.compareTo(f2));
}
}
```

Output:

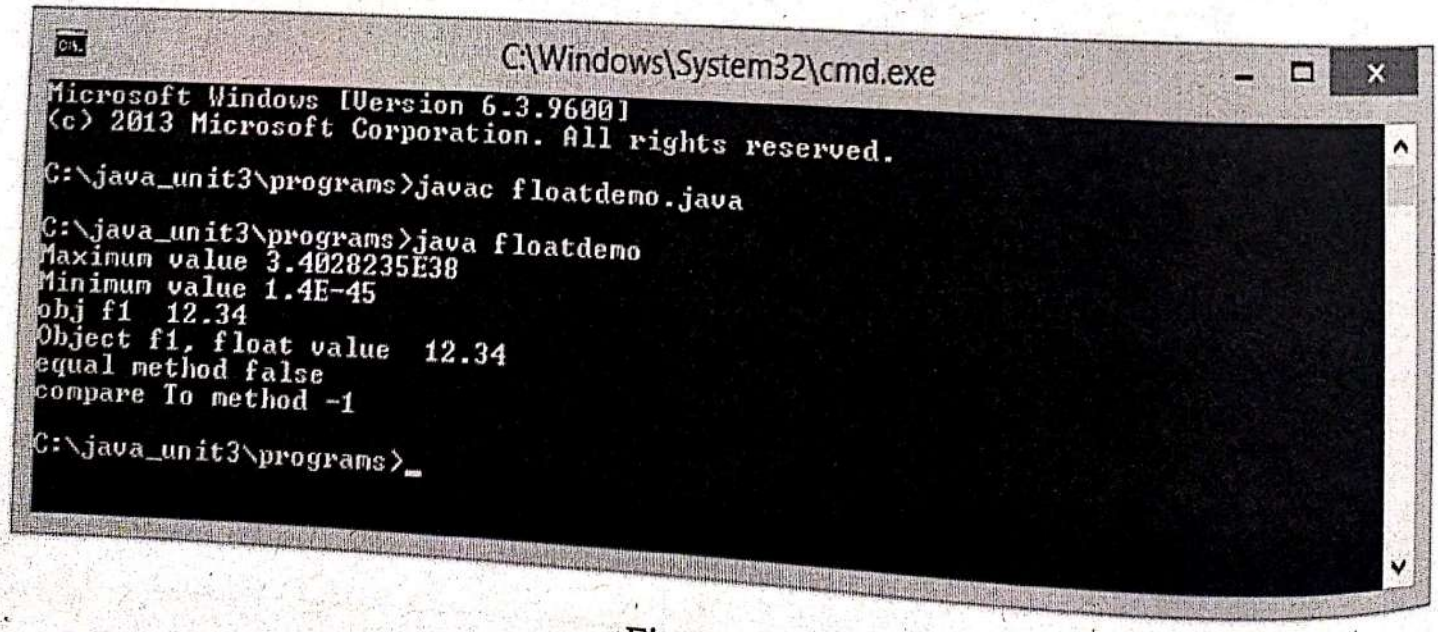


Figure 3.9

3.4.6 Double class:

Double class supports double primitive data type. This class provides many methods to support conversion of double to String, String to double and double to other primitive data types. With use of this class double data type can be converted to Double object. The constructors, methods and constants of Double class are described below:

Constructors of Double class:

Double(double value)

This constructor takes double primitive data type as an argument and generates Double class type object.

Double(String stringvalue)

This constructor construct Double type object from String type arguments.

Methods of Double class:

Method prototype	Method description
static int compare(double a, double b)	It compares two double type values numerically.
int compateTo(Double obj)	This method compares two Double objects and return the difference.
double doubleValue()	This method returns value of this Double as double.
static double parseDouble(String s)	This method takes String as an argument and convert it to double type value.
boolean equals(Object obj)	This method compares two Double objects and returns true if both are same otherwise it returns false.
static Double valueOf(double d)	This method returns Double object from the specified double value.
static Double valueOf(String s)	This method returns Double object from the specified String value.

Constants:

MIN_VALUE

Public static final double MIN_VALUE
This is constant minimum value a double can store. 2^{-1074}

MAX_VALUE

Public static final double MAX_VALUE
This is constant maximum value a double can store. $(2-2^{-52})\cdot 2^{1023}$

Program:

```
class Doubledemo
{
public static void main(String args[])
{
  Double d1= new Double(123.345);
  Double d2= new Double("4577.6777");

 System .out.println("Maximum value "+Double.MAX_VALUE);
System.out.println("Minimum value " +Double.MIN_VALUE);

  System.out.println("obj d1  "+ d1);
 System.out.println("Object d1, double value  " +d1.doubleValue());
 System.out.println("equal method "+d1.equals(d2));
 System.out.println("compare To method "+d1.compareTo(d2));
 }
}
```

Output:

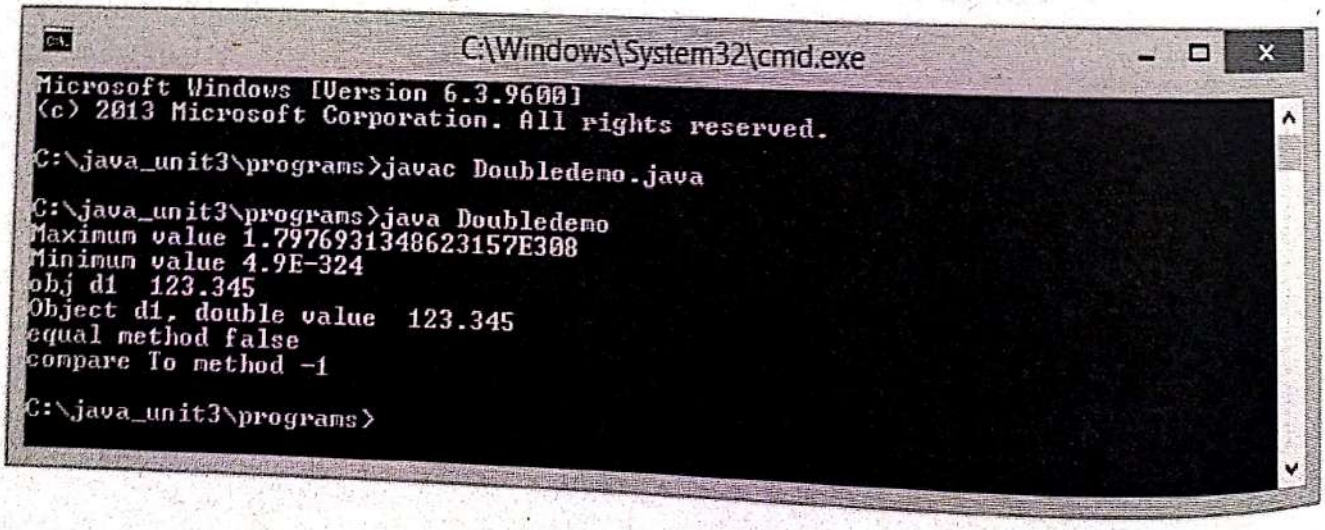


Figure 3.10

3.4.7 Boolean class:

Boolean class supports boolean primitive data type. This class provides many methods to support conversion of boolean to String, String to boolean and boolean to other primitive data types. With use of this class boolean data type can be converted to boolean object. The constructors, methods and constants of boolean class are described below:

Constructors of Boolean class:

Boolean(boolean value)

This constructor takes boolean primitive data type as an argument and generates Boolean class type object.

Boolean(String stringvalue)

This constructor construct Boolean type object from String type arguments.

Methods of Boolean class:

Method prototype	Method description
boolean booleanValue()	This method returns value of this Boolean as boolean.
static int compare(boolean a, boolean b)	It compares two boolean type values numerically.
int compateTo(Boolean obj)	This method compares two Boolean objects and return the difference.
static boolean parseBoolean(String s)	This method takes String as an argument and convert it to boolean type value.
boolean equals(Object obj)	This method compares two Boolean objects and returns true if both are same otherwise it returns false.
static Boolean valueOf(boolean b)	This method returns Boolean object from the specified boolean value.
static Boolean valueOf(String s)	This method returns Boolean object from the specified String value.

Constants:

TRUE

Public static final boolean TRUE
This is constant TRUE is associated to primitive value true.

FALSE

Public static final boolean FALSE
This is constant FALSE is associated to primitive value false.

Program:

```
class Booleandemo
{
public static void main(String args[])
{
  Boolean b1= new Boolean(true);
  Boolean b2= new Boolean("false");
   System.out.println("obj b1  "+ b1);
   System.out.println("equal method "+b1.equals(b2));
   System.out.println("compare To method "+b1.compareTo(b2));
   System.out.println("Parse Method  " + Boolean.parseBoolean("123"));
  }
}
```

Output:

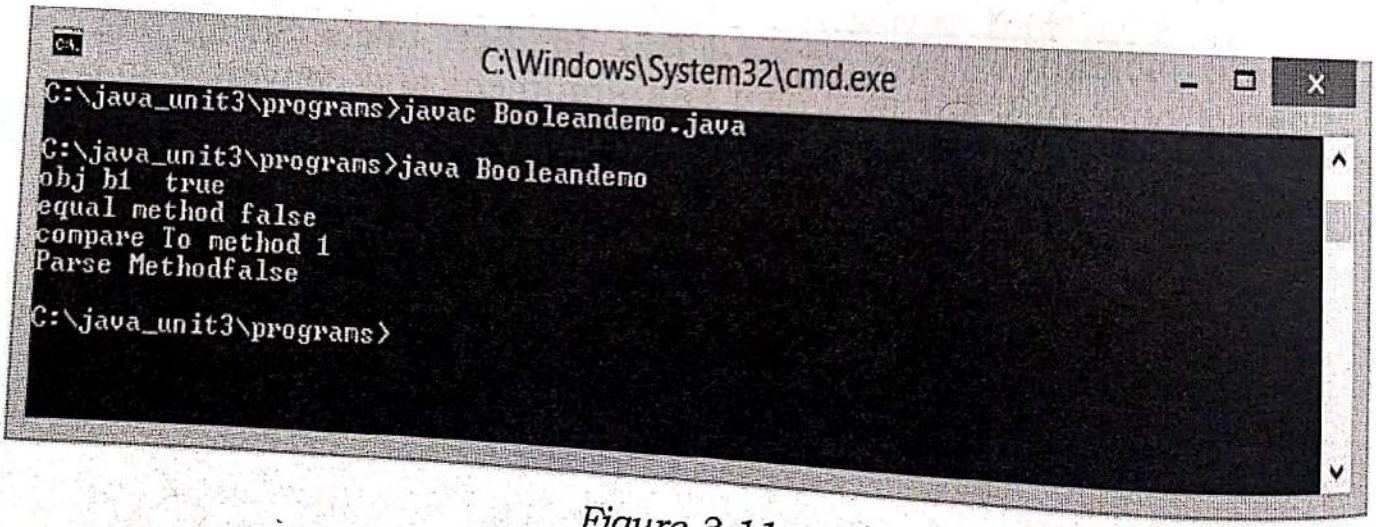


Figure 3.11

3.4.8 Character class:

Character class supports char primitive data type. This class provides myriad methods to identify the categories of character and convert the characters to uppercase to lowercase and vice versa. The constructors and methods are described below:

Constructor of Character class:

Character(char value)

This constructor takes char primitive data type as an argument and generates Character class type object.

Methods of Character class:

Method prototype	Method description
static int compare(char a, char b)	This method compare two character value numerically.
char charValue()	This method returns char value of this Character object.
int compareTo(Character obj)	This method compares two Character objects and return the difference.
boolean equals(Object obj)	This method compares two Character objects and returns true if both are same otherwise it returns false.
static boolean isDigit(char ch)	This method identifies the specified character is digit or not.
static boolean isLetter(char ch)	This method identifies the specified character is Letter or not.
static boolean isLetterOrDigit(char ch)	This method identifies the specified character is Letter or digit.
static boolean isLowerCase(char ch)	This method identifies the specified character is in lower case or not.
static boolean isUpperCase(char ch)	This method identifies the specified character is in upper case or not.
static char toLowerCase(char ch)	This method converts the specified character to lower case.
static char toUpperCase(char ch)	This method converts the specified character to upper case.
static Character valueOf(char ch)	This method returns Character object, representing the specified char value.

Program:

```
class Characterdemo
{
public static void main(String args[])
{
    Character obj1= new Character('A');
    Character obj2= new Character('B');

    System.out.println("equals method  "+obj1.equals(obj2));
    System.out.println("Compare to  method  "+obj1.compareTo(obj2));
    System.out.println("isDigit method  "+ Character.isDigit('1'));
    System.out.println("isLetter method  "+Character.isLetter('a'));
    System.out.println("isUpperCase method  "+ Character.isUpperCase('A'));
    System.out.println("toLowerCase method  "+Character.toLowerCase('D'));
    }
}
```

Output :

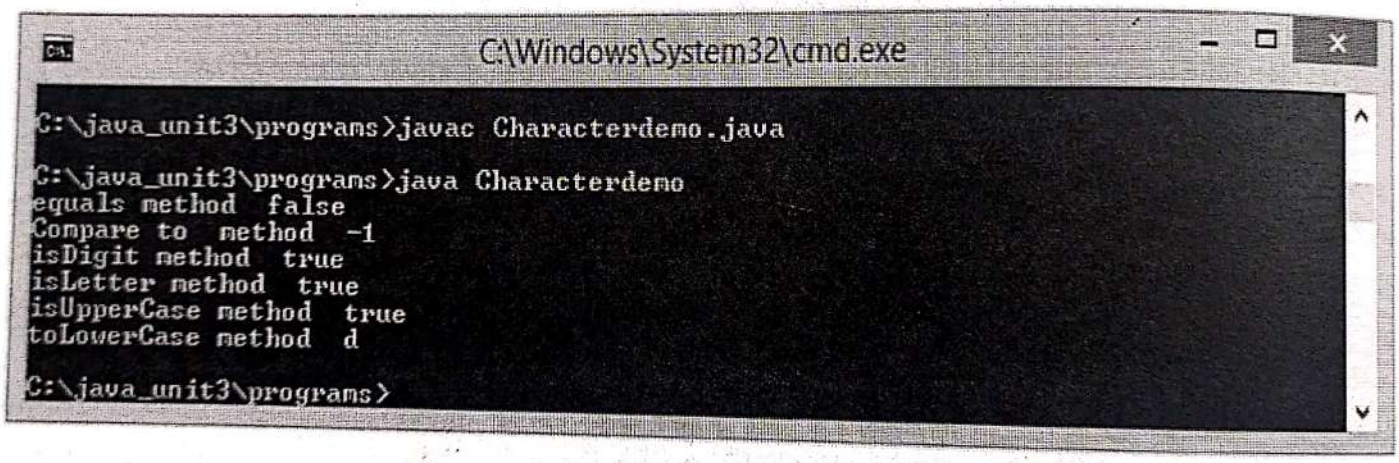


Figure 3.12

3.5 String class:

It is class available in java.lang package. It is very important class as it provides facilities to perform different operations on String. It provides numerous methods to work on string. It represents character strings and when we use string literals it is instance of this class. The strings are immutable. Once it is created, it can not be altered.

String s1="xyz";

char c1[]={'x','y','z'};

String s2= new String(c1);

Here string s1 and s2 both are same, in s1 we have created sting literal it is an instance of String class.

Constructors of String class:

String()

It represent empty string creation.

String(char ch[])

It represent string with the specified characters in the character array.

String(char ch[], int offset, int count)

It presents string subarray from character array argument.

Methods of String class:

Method Prototype	Method description
char charAt(int indexpostion)	This method returns the char value at the specified index position.
int compareTo(String s1)	This method compares two string and returns 0 if both are same otherwise returns non zero value.
int compareToIgnoreCase	This method compares two strings ignoring case differences.
String concate(String s1)	This method concatenates the specified String to the end of this string.
boolean endsWith(String suffix)	This method t ests if this string ends with the specified suffix.
boolean equals(Object obj1)	This method Compares this string to the specified object.
byte[] getBytes()	This method creates new byte array character from the String.
int indexOf(int ch)	This method returns the index within this string of the first occurrence of the specified character ch.
int indexOf(int ch, int fromindex)	This method returns the index within this string of the first occurrence of the specified character ch. It starts the search from the specific index position.
int indexOf(String str)	This method returns the index within

	this string of the first occurrence of the specified substring
int lastIndexOf(int ch)	This method returns the index within this string of the last occurrence of the specified character.
int lastIndexOf(String str)	This method returns the index within this string of the last occurrence of the specified substring.
int length	This method returns the length of this string.
String replace(char oldchar, char newchar)	This method returns the new String from replacing all old char to new char.
boolean stratsWith(String prefix)	This method tests if this string starts with the specified prefix.
char [] toCharArray()	This method converts this string to a new character array.
String to LowerCase()	This method converts all of the characters in this String to lower case.
String toUpperCase()	This method converts all of the characters in this String to upper case.

3.6 String Buffer class:

StringBuffer class also supports string operations. This class also provides many methods to work with string. The string created with string can not be modified but the string created with StringBuffer class can be modified. The String created with StringBuffer class can be modified and can grow in size.

The main task of StringBuffer are the insert and append methods. The append method insert the characters to the string buffer. The use of insert method is to add character at the specified position and the use of append method is add characters at the end of the string.

The basic difference between String class and StringBuffer class is, the capacity of StringBuffer class. The stringBuffer class is having internal buffer array it does not exceed the capacity. The internal buffer array allocated automatically. If it overflows, it is automatically made larger.

Constructors of StringBuffer class:

StringBuffer()

It creates a string buffer with no character in it and initial capacity is of 16 characters.

StringBuffer(int capacity)

This constructor construct a sting without any character and with initial specified capcacity.

StringBuffer(String str)

This constructor construct a StringBuffer from the specified String.

Methods of StringBuffer class:

Method Prototype	Method Description
StringBuffer append (boolean b1)	This method append the boolean argument to the string.
StringBuffer append (char c1)	This method append the character argument to the string.
StringBuffer append (char c1[])	This method append the character array argument to the string.
StringBuffer append (double d)	This method append the double argument to the string.
StringBuffer append (float f)	This method append the float argument to the string.
StringBuffer append (int i)	This method append the int argument to the string.
int capacity()	This method returns the current capacity of StringBuffer.
char charAt(int index)	This method returns character from the index position of invoking StringBuffer object.
void getChars(int sb, int se,char[]dst, int dstbgn)	This method copies characters from source string, from source begin to source end in the dst array from dstbgn position.
StringBuffer delete(int s, int e)	This method removes the characters from string buffer from start index to end index.
StringBuffer deleteCharAt(int index)	This method delete character from specified index position.
int indexOf(String str)	This method returns the index position of substring str from starting position.
Int indexOf(String str, int	This method returns the index

fromindex)	position of string str from mentioned, fromindex postion.
StringBuffer insert(int offset, char c)	This method insert the char c at offset positon of StringBuffer.
StringBuffer insert(int offset, char []s)	This method insert the char array s at offset positon of StringBuffer.
StringBuffer insert(int offset, float f1)	This method insert the float f1 at offset positon of StringBuffer.
StringBuffer insert(int offset, int i1)	This method insert the integer i1 at offset positon of StringBuffer.
StringBuffer insert(int offset, long l1)	This method insert the long l1 at offset positon of StringBuffer.
Int lastIndexOf(String str)	This method returns the last index of String str.
Int length()	This method returns the length of specified StringBuffer object.
String substring(int start)	This method returns the substring from the specified start index postion upto end.
String substring(int start, int end)	This method returns the substring from the specified start positon to the end postion.

3.7 Exception:

Exception handling mechanism of java, makes program robust. In this topic we will learn about exception handling in depth.

3.7.1 Exception introduction:

Exception is not an error. Exception is not logical error or syntax error but it is an abnormal condition when program can not work successfully. Exceptions can be handled and with use of it program can be gracefully terminated. There are many conditions when exception can be occurred in the program. The conditions are:

- Trying to write on the full disk.
- Trying to read from the file that does not exists.
- User has entered invalid data.
- Divide the number by zero.

This are called exceptions. That can handled and it is called exception handling. Error can not be handled. It must be solved.

To handle an exception java provides inbuilt base class called Exception that is available in java.lang package. The exception class hierarchy described below:

Exception class hierarchy.

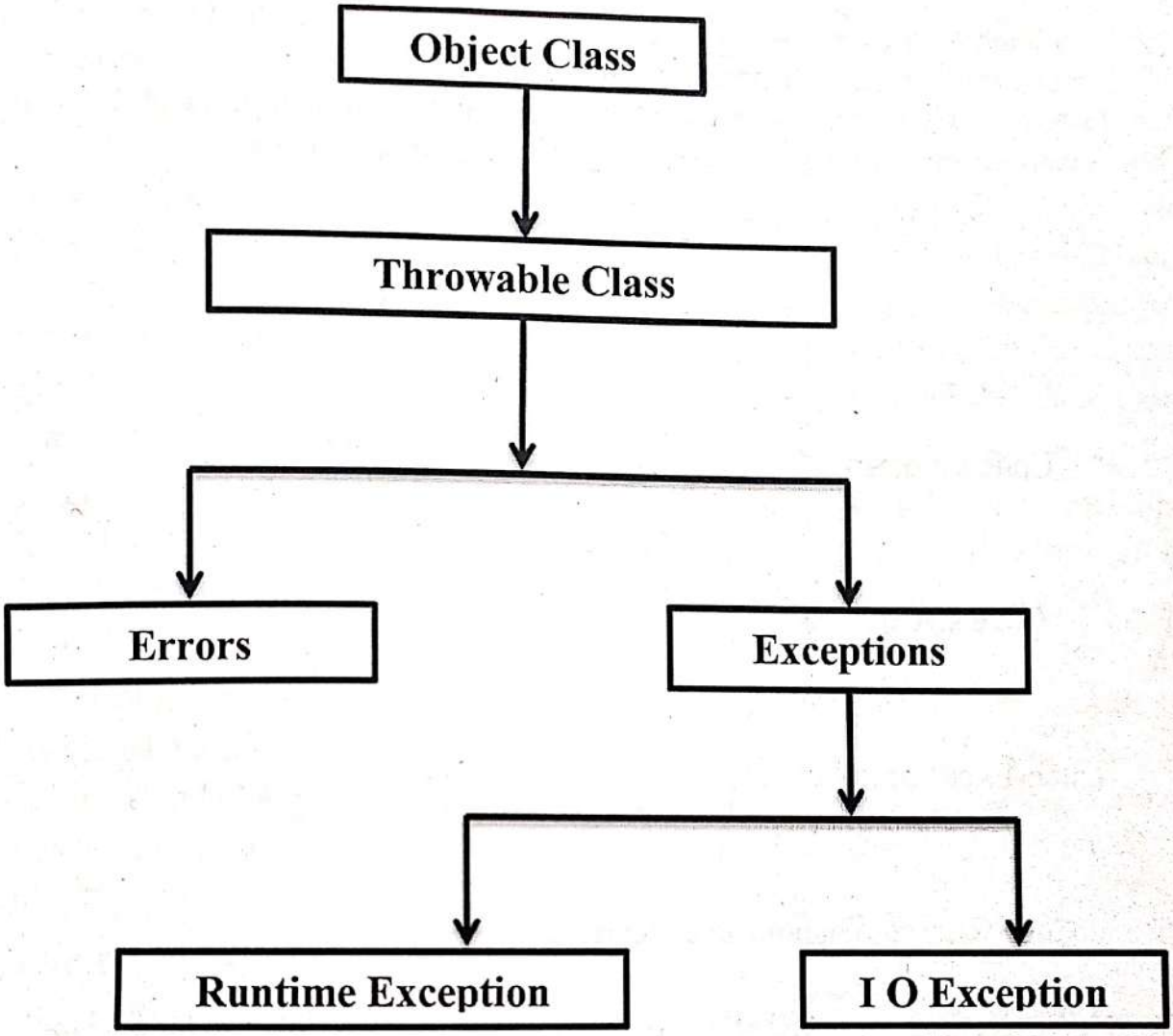


Figure 3.13

The figure describes Throwable is base class of all exception classes. From Throwable class Exception and Error class are inherited.

Exception class: it is base class of all exception classes. Exception class is further inherited in IOException and RuntimeException.

IOException: IOException occurred when Input Output related problems occurred.

RuntimeException: During program execution exceptions occurred are called RuntimeException. This exceptions can be handled. There are different RuntimeExceptions like, ArithmeticException, ArrayIndexOutOfBoundsException.

Error: Error is abnormal condition of the program. It can not be handled and it must be solved.

In java all exceptions are inbuilt classes and follows standard naming conventions.

3.7.2 Exception handling techniques:

To handle an exception java provides try, catch and finally blocks. The code, where exception can be occurred is written in try block. The try is keyword in java. When any exception occurred, what steps to be taken is written in catch block and catch is also keyword in java. The syntax is written below:

```
try
{
   Code statement1;
   Code statement 2;
   .....
   Code statement n;
}
catch(Exceptiontype1  e1)
{
  Code written to handle an exception;
}
catch(Exceptiontype2  e2)
{
  Code written to handle an exception;
}
finally
{
   Statements always executed;
}
```

As per the above syntax exception handling can be performed. Here the code where exceptions are possible is written within try block. If exception does not occurred then, the catch block will never executed. The catch is compulsory block that is always written after try block. Multiple catch blocks can be written after try block. It is possible that our single code may generate different types of exception, to handle this different types of exceptions multiple catch blocks can be written with different types of exception arguments. Here we have mentioned exception type in catch block and when any exception occurred the relevant catch block is executed. For example if Exceptiontype1 occurred the catch block of Exceptiontype1 is executed and if Exceptiontype2 occurred the catch block of Exceptiontype2 is executed. So at a time relevant catch block is executed.

When any exception occurred in try block the compile jump over the related catch block and the statement after it will never executed. As per the syntax if on statement2 exception occurred then from statement2 itself the compiler jump over the catch block and the next statement will never executed.

It may possible that due to exception handling mechanism certain required statements will never be executed. Like, suppose we have open the file in the beginning of try block, that should be closed at the and of it but before it if any exception occurred then the closing statement will never be executed and it resources will be blocked. To avoid such problem finally block is written.

3.7.3 The finally block:

The finally block is optional block. It is not compulsory to write finally after try and catch block but if you have written finally block then it will be always executed, either exception occurs or not. So certain statements that you always want to be executed can be written in finally block.

Now, let we learn exception handling through an example.

//Program to handle an exception:

```
class ExceptionDemo1
{
public static void main(String args[])
{
  int a=12, b=0,ans;
 try
{
 System.out.println("\n In try block");
```

```
    ans=a/b;
    System.out.println("Ans is" +ans);

  }
catch(ArithmeticException e1)
{
   System.out.println("In catch block - divide by zero exception occured");
}
finally
{
   System.out.println("In Finally block");
}
  }
}
```

Output:

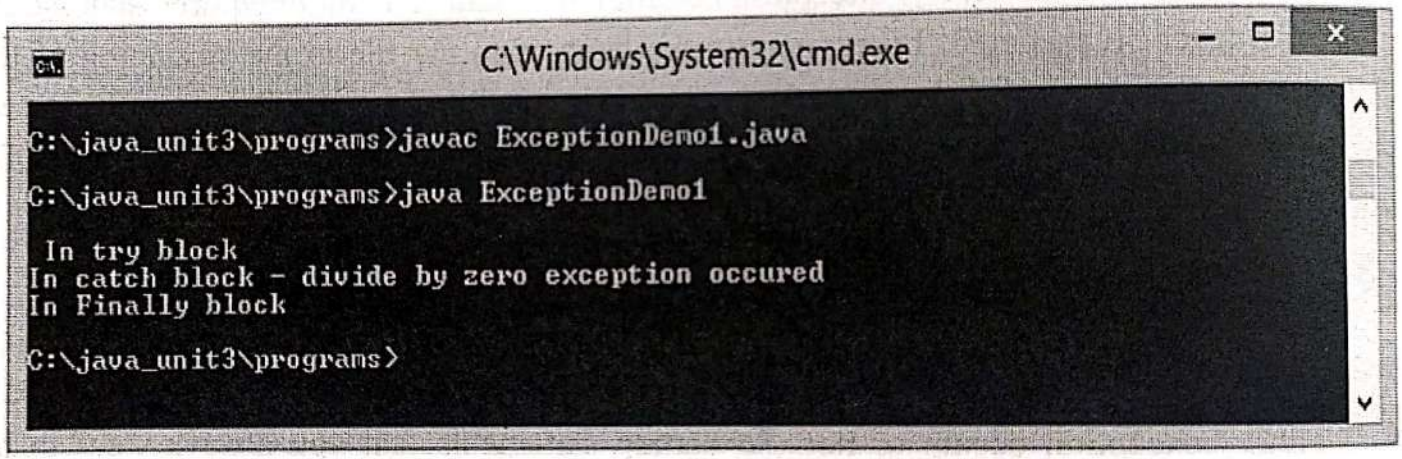


Figure 3.14

In above code we had written a/b, where the value of b is zero and exception occurred that is handled by catch block. The answer can not be printed because exception occurred before it.

Now let we change the code and find the output.

//Program to understand exception technique and finally block

```
class ExceptionDemo2
{
 public static void main(String args[])
 {
```

```
 int a=12, b=3,ans;
 try
{
 System.out.println("\n In try block");
  ans=a/b;
 System.out.println("Ans is" +ans);

}
catch(ArithmeticException e1)
{
  System.out.println("In catch block - divide by zero exception occured");
}
finally
{
 System.out.println("In Finally block");
}
 }
}
```

Output:

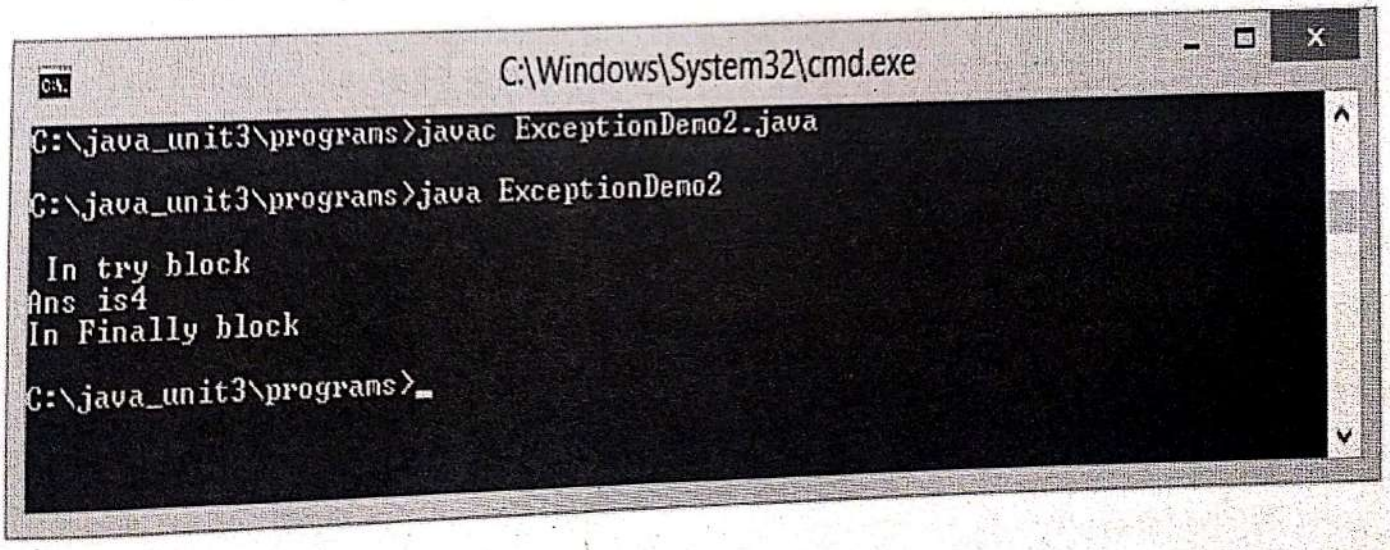


Figure 3.15

In above code the b value is 3 so, exception does not occurred and catch block will never executed and we can find the answer. The finally block executes always either exception occurs or not.

3.7.4 The catch block hierarchy:

The block always followed by catch block and we can write multiple catch block and relevant catch block can be executed when exception occurs. Let we learn from the following example.

//Program with multiple catch blocks

```
class ExceptionDemo3
{
 public static void main(String args[])
 {
  int a=12,ans;
  int b[ ] ={1,2,0,3};

  for(int i=0;i<5;i++)
  {
  try
  {
   ans=a/b[i];
   System.out.println("Ans is" +ans);
  }
  catch(ArithmeticException e1)
  {
    System.out.println("In catch block - divide by zero exception occured");
  }
  catch(ArrayIndexOutOfBoundsException e2)
  {
  System.out.println("In catch block - array index out of bounds exception occured");
  }
  catch(Exception e3)
  {
  System.out.println("In catch block - Exception class");
  }
  }
  }
}
```

Output:

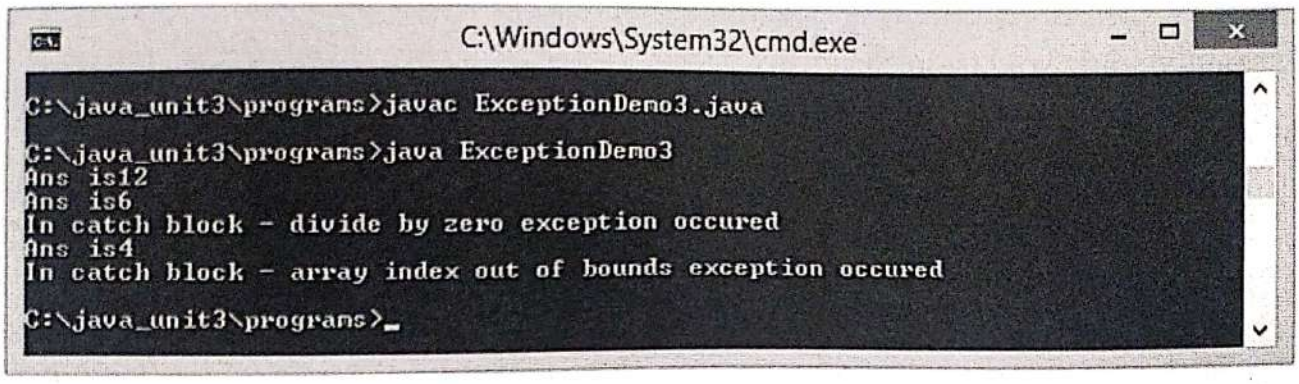


Figure 3.16

In the above program we have used multiple catch block and relevant catch block is executed when any exception occurs. Here we have written catch(Exception e3) after two previous catch block. When we write multiple catch blocks after try we have to be careful about catch block hierarchy. Exception class should be written at last because it is base class of all Exception classes and able to handle all kind of exception, if we have written it before other classes then, the following catch block will become unreachable code.

3.7.5 The throw and throws keywords:

The 'throw' is important key word of exception handling mechanism. It is used to throw any exception explicitly. This keyword is generally used to throw user defined exceptions. Only Throwable class or its sub classes can be use 'throw' keyword.

The 'throws' keyword is used to broadcasting. If any method can throw any kind of exception then with the signature of that method throws keyword is used so the caller of the method can handle it appropriately.

3.7.6 User Defined Exceptions:

User defined exception is also called, custom exception. Java provides mechanism to programmer, to create their own exceptions as per the requirements of the program. For example suppose we want to create an exception where the user enters string from command line argument and the string should be "java", if the string is not "java" then custom exception can be thrown. Let we learn through an example.

//Program of custom exception

```
class StringInvalid extends Exception
{
 StringInvalid(String s)
  {
  System.out.println("String is not valid exception " +s);
  }
}
```

```
class Myexp
{
 public static void main(String args[])
 {
  String s=args[0];
  if(s.equals("java"))
   System.out.println("Valid");
  else
   {
   try{
      throw new StringInvalid(s);
   }
catch(Exception e)
   {
   System.out.println("Exception handled");
  }
  }
}
}
```

Output:

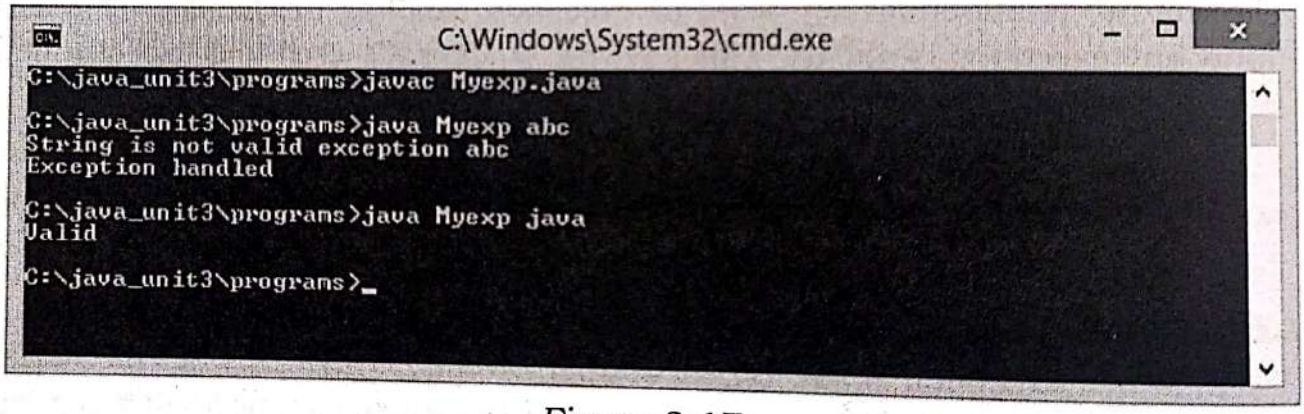


Figure 3.17

Here StringInvalid is custom exception, it is subclass of Exception class. in Myexp class string is entered through command line arguments and if the string is "java" then valid message is printed otherwise stringInvalid exception thrown that invokes the same class's constructor and the catch block handle the exception.

In conclusion we have learn about package, creation of user defined package, access specification concerned to package. We have also learned about lang package, Object class and wrapper classes. We have learned about String and StringBuffer class, Exception handling mechanism and user defined exception.

☺ ☺ ☺ ☺ ☺

Exercises

Answer the following:

1. What is package? Write steps to create package and use the package.
2. Which are different types of package?
3. Explain access protection with use of the package.
4. Explain java.lang package.
5. Explain Object class.
6. What is wrapper class? Explain any two wrapper classes.
7. Explain byte and short wrapper class.
8. Explain any five methods of integer wrapper class.
9. Explain any five methods of float wrapper class.
10. Explain Long class with constructors and methods.
11. Explain Double class with constructor and any five methods.
12. Write difference between String and StrinBuffer class.
13. Explain any five methods from String class.
14. Explain any five methods from StrinBuffer class.
15. What is exception? Explain exception class hierarchy.
16. Explain exception handling mechanism.
17. Explain use of finally block.
18. What is user defined exception? Write steps to create user defined exception.
19. Explain hidden class.
20. Explain boxing and unboxing.

True or False / Multiple choice Question-Answer:

1. To use package in our program _____ keyword is used.
   a. import
   b. include
   c. Any of the above
   d. None of the above

2. The physical structure of java files must _____ as mentioned in the package.
   a. Same
   b. Different
   c. Not decided
   d. Any of the above

3. To use all classes from awt package _____ syntax should be used.
   a. import java.awt.*;

b. import java.awt;
c. include java.awt.*;
d. include java.awt;

4. With use of package name clashes can be avoided.
   a. True
   b. False

5. Java support _____ types of packages
   a. User defined
   b. Inbuilt
   c. Both of the above
   d. None of the above

6. To create class under any package, _____ should be the first statement.
   a. package
   b. import
   c. Any of the above
   d. None of the above

7. If in package single java file is having more than one classes then only one class can be public.
   a. True
   b. False

8. Protected access specification allows access to all classes in the same package and sub classes in the any package.
   a. True
   b. False

9. Public access specification allows access at any where in the same package or any other package.
   a. True
   b. False

10. _____ Package provides fundamental classes for java language.
    a. lang
    b. awt
    c. applet
    d. language

11. _____ is the base class of all java classes.
    a. Object
    b. Java

c. Language
d. None of the above

12. The wrapper class for byte data type is _____.
    a. Byte
    b. Short
    c. Integer
    d. Long

13. The wrapper class for boolean data type is _____.
    a. Boolean
    b. Bool
    c. Byte
    d. Short

14. To convert string to byte data type _____ method is used.
    a. parseByte
    b. stringtoByte
    c. valueOf
    d. none of the above

15. To compare two integer objects _____ method is used.
    a. equals
    b. equal
    c. compare
    d. compareObject

16. To compare two short value numerically _____ method is used.
    a. compare
    b. compareTo
    c. compareShort
    d. Any of the above

17. Integer value can be converted to binary string with _____ method.
    a. toBinaryString
    b. convertTobinary
    c. intTobinary
    d. None of the above

18. _____ method identifies that the character is letter or not.
    a. isLetter
    b. toLetter
    c. convertToLetter

d. None of the above

19. _____ method converts the character to upper letter.
    a. toUpperCase
    b. toupperLetter
    c. inupper letter
    d. None of the above

20. To create byte array from string _____ method is used.
    a. getBytes
    b. get_bytes
    c. putbytes
    d. None of the above

21. String created with _____ class can be modified.
    a. String
    b. StringBuffer
    c. Any of the above
    d. None of the above

22. _____ is base class of all Exception classes.
    a. Throwable
    b. RuntimeException
    c. IOException
    d. None of the above

23. In exception handling mechanism _____ code will always executed.
    a. catch
    b. finally
    c. Any of the above
    d. None of the above

24. _____ keyword is used to throw an exception explicitly.
    a. try
    b. catch
    c. throw
    d. throws

25. _____ keyword is used to broadcast an exception.
    a. try
    b. catch
    c. throw
    d. throws

MCQ Solution:

1 – a	2 – a	3 – a	4 – a	5 – c
6 – a	7 – a	8 – a	9 – a	10 – a
11 – a	12 – a	13 – a	14 – a	15 – a
16 – a	17 – a	18 – a	19 – a	20 – a
21 – b	22 – a	23 – b	24 – c	25 – d

CC-214 Core Java Practical (UNIT – 3)

1. Create a package P and within that package create class PackClass which have method called findmax( ) which find maximum value from three numbers. Now import the package within another class DemoClass and now display the maximum number.

```
// class pack from package p

package p;
public class pack
{
   public int findmax(int a, int b,int c)
   {
     int max=((a>b)?((a>c)?a:c):((b>c)?b:c));
     return(max);
   } .

}
// demo class

import p.*;
class demo
{
  public static void main(String args[])
{
  pack p1= new pack();
  int ans=p1.findmax(10,5,23);
  System.out.println("Answer is"+ans);
```

```
  }

}
```

2. Write a program that creates three different classes in three different packages and access them from default package. All the three packages should be at the same level.

```
//package p1
package p1;
public class c1
{
  public void printc1()
  {
    System.out.println("Package p1 and class c1");
  }

}
//package p2
package p2;
public class c2
{
  public void printc2()
  {
    System.out.println("Package p2 and class c2");
  }

}
//package p3
package p3;
public class c3
{
  public void printc3()
  {
    System.out.println("Package p3 and class c3");
  }

}
//package access
package access;
```

```
import p1.*;
import p2.*;
import p3.*;
public class acs
{
  public static void main(String args[ ])
  {
    c1 obj1 = new c1();
    c2 obj2 = new c2();
    c3 obj3 = new c3();
    obj1.printc1();
   obj2.printc2();
    obj3.printc3();

  }

}
```

3. **Create package pack1 within this package create class A which contains one instance variable and one instance method. Create another package pack2 within this package create class B. where class B is calling the method and variable of class A**

```
//package pack1

package pack1;
public class A
{
 public int a=10;
  public void printA( )
  {
    System.out.println("Package pack1, class A and method printA");
  }

//package pack2
package pack2;
import pack1.*;

class B
```

```
{
  public static void main(String args[])
 {
      A a1= new A();
     System.out.println("\n Instance variable "+a1.a);
     System.out.println("\n Instance method calling");
     a1.printA();

  }
}
```

4. Write a program that accepts a string from command line and perform following operations:

1. **Display each character on separate line in reverse order.**
2. **Count total number of chracters and display each character's position too.**
3. **Identify that whether the string is palindrom or not.**
4. **Count total number of uppercase and lowercase characters in it.**

```
class u3_p4

{
 public static void main(String args[])
 {
  char c[ ]= args[0].toCharArray();

  System.out.println("\n Display each character on seperate line in reverse order");

  for(int i=c.length-1; i>=0; i--)
    System.out.println(c[i]);

 System.out.println("\n Count total number of character and display each character's position");
 System.out.println("\n Total number of characters are " + c.length);
 for(int i=0; i<c.length;i++)
  System.out.println(c[i]+ "  character is at " + i + " position");


System.out.println("\n Identify the string is palindrom or not");
StringBuffer s1= new StringBuffer(args[0]);
```

```
 StringBuffer s2= s1.reverse();
if(s1.equals(s2))
 System.out.println("\n String is palindrom");
else
 System.out.println("\n String is not palindrom");

System.out.println("\n Total number of uppercase and lower case characters");
int up=0,lw=0;
for(int i=0; i<c.length;i++)
{
  if(Character.isUpperCase(c[i]))
     up++;
  else
    lw++;

 }

 System.out.println("\n Uppercase characters are: "+up);
 System.out.println("\n Lowercase characters are: "+lw);
}
}
```

5. Write a Java program to input n integer numbers and display lowest and second lowest number. Also handle the different exceptions possible to be thrown during execution.

```
class Prog2
{
  public static void main(String[] args)
 {
    int a[]= new int[args.length];
    int temp,t;
  try {
        for(int i=0;i<args.length;i++)
        {
          a[i]= Integer.parseInt(args[i]);
        }
       for(int i=0;i<args.length;i++)
        {
          for(int j=i+1;j<args.length;j++)
```

```
            {
                if(a[i]>a[j])
                {
                    temp=a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
              //  t= a[i]/a[j];
            }
        }
     System.out.println("Lowest" + a[0]+"Second Lowest"+a[1]);
    }
     catch (ArithmeticException e)
    {
       System.out.println("ArithmeticException occured");
    }
     catch (ArrayIndexOutOfBoundsException e)
      {
      System.out.println("ArrayIndexOutOfBoundsException                "+"Enter
Command Line arguments"); }
      catch (NumberFormatException e) {
       System.out.println("NumberFormatException  "+"Enter Numbers only"); }
catch (Exception e) {
System.out.println(""+e.getMessage()); } } }
```

6. Write a program that takes a string from the user and validate it. The string should be at least 5 characters and should contain at least one digit. Display an appropriate valid message.

```
class validatestring extends Exception
{
   validatestring(String s)
   {
    System.out.println("\n String is not valid  "+s);
   }
}
class u3_p6
{
 public static void main(String args[])
 {
    String s= args[0];
```

```
    char c[]= s.toCharArray();
    int flag=0;
   try{
   if(s.length()>=5)
   {
      for(int i=0;i<s.length();i++)
       {
          if(Character.isDigit(c[i]))
           {
           flag=1;
           System.out.println("\n String is valid");
           break;
           }
       }
       if(flag==0)
         throw new validatestring(s);
    }
   else
    {
      throw new validatestring(s);
    }
   }catch(Exception e){}
}

}
```

7. Write an application that accepts marks of three different subject from user. Marks should be between 0 to 100, if marks of any of the subject is not belong to this range, generate custom exception out of RangeException. If marks of each subjects are greater than or equal to 40 then display message "PASS" along with percentage, otherwise display message "FAIL". Also write exception handling code to catch all the possible runtime exceptions likely to be generated in the program.

```
class RangeException extends Exception
{
   RangeException(int i)
   {
      super(" RangeException:  Marks is not valid  "+i);
   }
```

```
}
class u3_p7
{
 public static void main(String args[])
 {
   int a[] = new int[3];
   int sum=0;
   float per=0.0f;
  for(int i=0;i<3;i++)
   {
          try{
           a[i]= Integer.parseInt(args[i]);
           if(a[i] <0 || a[i]>100)
           {
              throw new RangeException(a[i]);
           }
           else if(a[i]>=40)
           {
              sum=sum+a[i];
              System.out.println("\n Pass in subject "+i);
           }
           else
           {
              sum=sum+a[i];
              System.out.println("\n Fail in subject "+i);

           }

          }
        catch(NumberFormatException e) {
                System.out.println("Number Format Exception"); }
        catch(ArithmeticException e) {
                System.out.println("arithmetic exception occur"); }
        catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("array index exception occur"); }
        catch(Exception e) {
                System.out.println("Exception: "+e.getMessage());
                }

    }
    per=(float) sum/3;
```

```
System.out.println("\n Percentage is "+per);

}

}
```

8. Write a program which takes the age of 5 persons from command line and find the average age of all persons. The program should handle exception if the argument is not correctly formatted and custom exception if the age is not between 1 to 100.

```
class RangeException extends Exception {
RangeException(String message) {
super(message); } }
class TestMyException {
public static void main(String args[]) {
int a[]=new int[5];
try {
for(int j=0;j<5;j++) {
a[j]=Integer.parseInt(args[j]);
if(a[j]>100||a[j]<0)
throw new
RangeException("invalid Range"+a[j]);} }

catch(NumberFormatException e) {
System.out.println("Number Format Exception"); }
catch(ArithmeticException e) {
System.out.println("arithmetic exception occur"); }
catch(ArrayIndexOutOfBoundsException e) {
System.out.println("array index exception occur"); }
catch(Exception e) {
System.out.println("Exception"+e.getMessage()); } } }
```

9. Write an application that converts between meters and feet. Its first commandline argument is a number and second command line argument is either "centimeter" or "meter". If the argument equals "centimeter" displays a string reporting the equivalent number of meters. If this argument equals "meters", display a string reporting the equivalent number of centimeter. If unit is not given properly then generate custom exception Unitformatexception. If first argument is not proper format then

generate numberformatexception. Generate other exception as per requirements. (1 meter=100 centimeter)

```
class UnitformatException extends Exception
{
   UnitformatException(String s)
   {
      super(" UnitformatException: unit is not valid  "+s);
   }
}
class u3_p9
{
 public static void main(String args[])
 {
    int no;
    String u;
          try{
            no= Integer.parseInt(args[0]);
            u=args[1];
            if((u.equals("centimeter")) || (u.equals("meter")))
             {
                 if(u.equals("centimeter"))
                  {
                    int m=no/100;
                    System.out.println("Equivalent number of meter is " +m);
                  }
                 else
                  {
                    int cm=no*100;
                    System.out.println("Equivalent number of centi meter is " +cm);
                  }
              }
            else
              throw new UnitformatException(u);

           }
         catch(NumberFormatException e) {
                 System.out.println("Number Format Exception"); }
         catch(ArithmeticException e) {
                 System.out.println("arithmetic exception occur"); }
         catch(ArrayIndexOutOfBoundsException e) {
```

```
                System.out.println("array index exception occur"); }
        catch(Exception e) {
                System.out.println("Exception: "+e.getMessage());
                }

    }
```

10. Write a program that accepts 5 even numbers from command line , if any of the numbers is odd then throw custom exception OddException and count such invalid numbers.

```
class Oddnumber extends Exception
{
 Oddnumber(int i)
 {
  System.out.println("Number is odd " +i);
 }
}
class myexp4
{
 public static void main(String args[])
 {
  int a[]= new int[args.length];
 for (int i=0;i<args.length;i++)
 {
 try{
    a[i]=Integer.parseInt(args[i]);
    if(a[i]%2 !=0)
```

```
            throw new Oddnumber(a[i]);
        }
        catch(Exception e)
        {
          System.out.println("Exception occured");
        }
      }
    }
}
```

Unit - 4

Multithreading and Applet

- MULTITHREADING
- APPLET

Unit-4 Multithreading and Applet

4.1 MULTITHREADING:

4.1.1 Introduction:

A program can be divided into a number of small processes. Each small process can be addressed as a single thread (a lightweight process). A thread is a single sequential flow of control within a program. Multithreaded programs contain two or more threads that can run concurrently and each thread defines a separate path of execution. This means that a single program can perform two or more tasks simultaneously. For example, one thread is writing content on a file at the same time another thread is performing spelling check. In other words, a single program having multiple threads, executing concurrently, is known as multithreaded program.

Today's, computers are multitasking systems. Working on more than one task at the same time is known as multitasking. UNIX, LINUX and Windows are some example of multitasking systems. Java achieved multitasking concept using Thread. Multithreading is a form of multitasking. There are two types of multitasking.
Process-Based Multitasking :(also called Multiprocessing)

- Here, programs are considered as processes. Two or more processes or programs can be run concurrently.
- Process is a considered as heavy-weight task.
- Process requires its own address space
- Context switching of CPU from one process to another requires more overheads.

Thread-Based Multitasking :(called Multithreading)

- It is executing a program, with more than one threads, each thread performing different tasks simultaneously.
- Thread is considered as light-weight task.
- Threads share same address space.
- Context switching of CPU occurs within the program hence requires less overheads. Multithreading allows gain access over idle time of CPU. Hence, CPU idle time is minimized using multithreading.

4.1.2 Multithreading in JAVA:

JAVA is a multi-threaded programming language. Every program in JAVA has at least one thread, i.e. *main* thread. When we run a JAVA program, the program code starts its execution from main() method. Therefore, the JVM creates a thread to start executing the code present in **main** method. This thread is called as **main** thread. A

thread can die naturally or can be forced to die. A thread dies naturally when it exits *run()* method normally. A thread can be killed or interrupted by calling *interrupt()* method.

4.1.3 java.lang.Thread:

The Thread class is available in java.lang package. By default, this package is implemented in every Java program. It is declared as,
public class Thread
extends Object
implements Runnable

To create a thread using Thread class, we create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its execution inside run() method. By creating an object of new class ,we can call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object. The constructors of Thread class are :

- Thread() : Constructor without any arguments, which means it uses the default values.
- Thread(String : threadname): The name of the constructor can be specified as String
- Thread(ThreadGroupthreadgroup, String threadname) : We can specify the thread group and thread name.

Thread class defines many methods of managing threads. Some of them are as follows

Methods of Thread class

Methods	Description
void start()	Starts a thread by calling its run() method.
void run()	Thread begins its execution from this method. Entry point of a thread.
static void sleep(long ms)	Suspends a thread for specified period of milliseconds(ms).
final String getName()	Returns the threads name.
final void setName(String threadname)	Sets the threads name to threadname.
final void join()	Waits for a thread to terminate.
final boolean isAlive()	Checks if thread is still running or not and returns a Boolean value.
void interrupts()	Interrupts a thread
static void yield()	The current executing thread is paused and allows other threads to execute.

final void setPriority(int priority)	Set new priority of thread .
final int getPriority()	Returns the priority of the thread.
static Thread currentThread()	Returns the reference to the currently executing thread.

4.1.4 The Main Thread:

Every Java program has at least one thread, the main thread.
When a java program starts execution, the JVM creates the main thread and calls the main() method from within that thread. There are two main points to remember of main thread

- The main thread spawns the other threads. These spawned threads are called child threads.
- The main thread is always last to finish its execution as it is responsible for releasing all the resources used by the program during its execution.

The main thread is created automatically when our program is started. To control it we must obtain a reference to it. This can be done by calling the method currentThread( ) which is present in Thread class. This method returns a reference to the thread on which it is called. The default priority of Main thread is 5 and for all remaining user threads priority will be inherited from parent to child.

Example 1: Program showing demo of currentThread(), setName() and getName() method of Thread class

```
class mainthread
  {
public static void main(String args[])
      {
         Thread th = Thread.currentThread();
System.out.println("Current thread : " + th);
System.out.println("Name of the current thread : "+th.getName());
th.setName("Mythread");
System.out.println("Thread after renaming : " + th);
   System.out.println("Name of the current thread : "+th.getName());
      }
}
```

Output:

```
D:\JAVA>javac mainthread.java

D:\JAVA>java mainthread
Current thread : Thread[main,5,main]
Name of the current thread : main
Thread after renaming : Thread[Mythread,5,main]
Name of the current thread : Mythread

D:\JAVA>
```

4.1.5 Creation of New Threads:

There are two ways of creating a thread in java.

1. By extending the **Thread class**.
2. By implementing the **Runnable interface**.

➢ **Creating threads by extending Thread class**

Threads can be created by extending java.lang.Thread class. To create a thread :

1. We have to create a new class that extends Thread class.
2. Then override the run() method by writing code required for thread.
3. And then to create an instance of that class. The run() method is what is executed by the thread after you call start() method.

Example 2: Program to create a thread using Thread class.

```
classnewthread extends Thread
{    public void run() {
                for(int i=0;i<5;i++)
                        System.out.println("Thread demo extending thread class"); }
}
classthread_demo
{
public static void main(String args[])
{
        newthread  a = new newthread();
        Thread th = new Thread(a, "My_Thread");
        th.start();  // start() method calls run() method of class newthread
        }
}
```

Output:

```
D:\JAVA>javac thread_demo.java

D:\JAVA>java thread_demo
Thread demo extending thread class
Thread demo extending thread class
Thread demo extending thread class
Thread demo extending thread class
Thread demo extending thread class

D:\JAVA>_
```

- **Creating threads by implementing Runnable interface**

Another way of creating a thread id by implementing Runnable interface. This class provides only one method:
public void run()

This method is basically similar to method of Thread class, which is used to write code required for threads. This interface is declared public as :

public interface Runnable

The interface is to be implemented by any class whose instance is to be executed by a thread. 'implements' keyword is used to implement Runnable interface.

Example 3: Program to create a thread using Runnable interface

```
class myclass implements Runnable
{    public void run()
{
                for(int i=0;i<5;i++)
                        System.out.println("Thread demo using Runnable");
 }
}
class thread_runnable
{ public static void main(String args[])
{
Thread t1 = new Thread(new myclass ());
t1.start();
}
}
```

Output:

```
D:\JAVA>javac thread_runnable.java

D:\JAVA>java thread_runnable
Thread demo using Runnable
Thread demo using Runnable
Thread demo using Runnable
Thread demo using Runnable
Thread demo using Runnable

D:\JAVA>
```

Example 4: Program to demonstrate multithreading. Program should create two threads. First thread should find sum of first five numbers. Second thread should find product of first five numbers.

```
class sum_thread extends Thread
{
  int sum=0;
    public void run()
     {
        for(int i=1;i<=5;i++)
              {
                    sum = sum + i;
                    System.out.println("Sum = " +sum);
              }
       }
}
class mul_thread extends Thread
{
  int mul =1;
public void run()
{
        for(int i=1;i<=5;i++)
              {
                    mul = mul * i;
                    System.out.println("multiplication = " + mul);
              }
       }
```

```
}
class multithreaddemo
{
  public static void main(String args[])
{
        sum_thread st = new sum_thread();
        Thread s_th = new Thread(st,"Sum Thread");
        .out.println("Name of the thread = "+s_th.currentThread());

        mul_thread mt = new mul_thread();
         Thread m_th = new Thread(mt,"Sum Thread");
        System.out.println("Name of the thread = " + m_th.currentThread());

         s_th.start();
        m_th.start();
}
}
```

Output:

```
D:\JAVA>javac multithreaddemo.java

D:\JAVA>java multithreaddemo
Name of the thread = Thread[main,5,main]
Name of the thread = Thread[main,5,main]
Sum = 1
Sum = 3
Sum = 6
multiplication = 1
Sum = 10
multiplication = 2
Sum = 15
multiplication = 6
multiplication = 24
multiplication = 120

D:\JAVA>_
```

A thread passes through several states throughout its life time. Hence, it is also known as Life Cycle of Thread. Possible states of Thread.state.

- New
- Runnable
- Not Runnable
  - Waiting/Blocked
  - Timed waiting
- Terminated / Dead

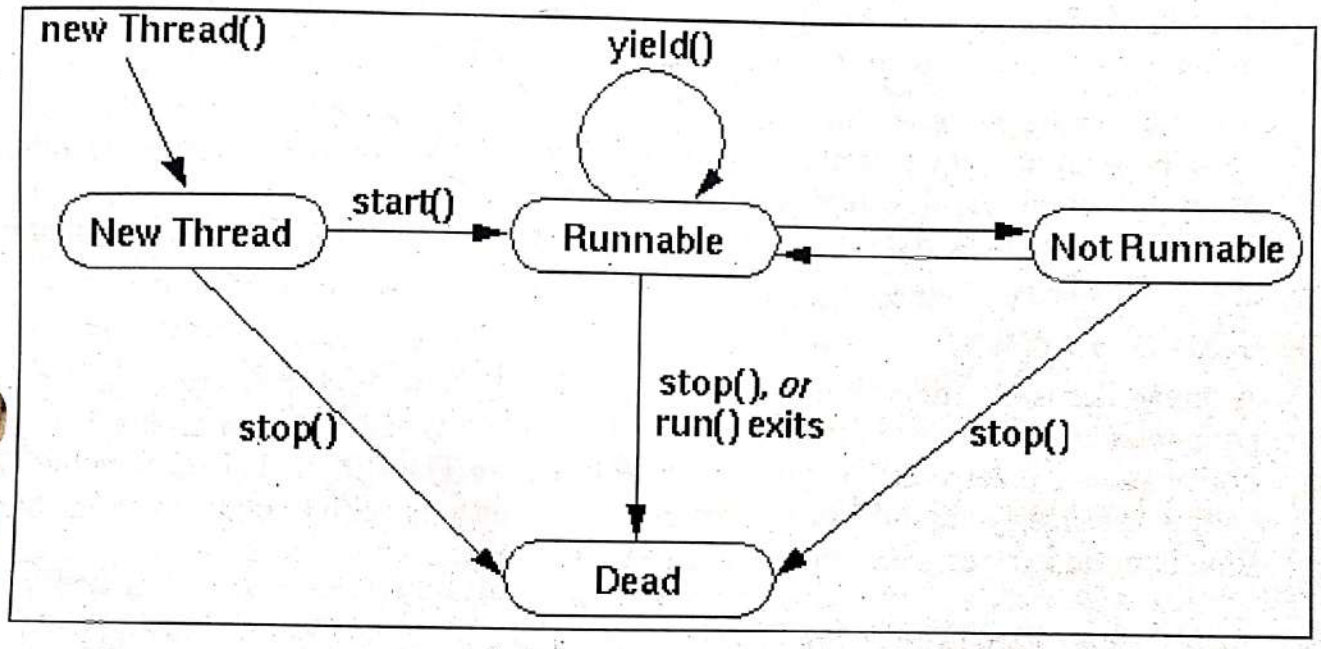


Figure : Thread States

1. NEW:

In NEW state, a thread is created , but not started. When a thread lies in the new state, it's code is yet to be run and hasn't started to execute. From this state either a thread can be started by calling start() method or can be stopped using interrupt() method. No other method apart from start() and interrupt() can be called from this state.

2. RUNNABLE:

Runnable state is also known as Ready-to-run state. From New state , the thread might move to Runnable state. Calling start() method on thread puts it in Runnable state. In this state, a thread might actually be running or it might be ready run at any instant of time. A running thread is one which is currently under execution by the processor. It can be called using currentThread() method. A runnable thread is one which is ready to run ,and waiting in queue for processor. When thread is in running state, the instructions of run() method are executed.

3. NOT RUNNABLE:

From Runnable state , a thread might move to Not Runnable state. Not Runnable is a hypothetical state, used to categorize following states:
Waiting/Blocked and Timed waiting states

- **WAITING/BLOCKED**

When a thread is not running, then it is either Blocked or Waiting

For example, when a thread is waiting for I/O to complete, it lies in the blocked state. Then, the thread scheduler should reactivate and schedule a blocked/waiting thread. A thread in this state cannot continue its execution any further until it is moved to runnable state. Any thread in these states does not consume any CPU cycle.

A thread can be put in waiting state for various reasons e.g. calling it's wait() method. Usually program put a thread in WAIT state because something else needs to be done prior to what current thread is doing.
Once the thread wait state is over or , it's state is changed to RUNNABLE ,so that it can resume its execution.

- **TIMED WAITING**

A thread lies is in Timed Waiting state when it calls a method with a time out parameter. A thread lies in this state until the timeout is completed or until a notification is received. You can put a java thread in TIMED WAITING state by calling it's sleep(long milliseconds) method or wait(long milliseconds) method. After time interval expires, such threads return to Runnable state

4. TERMINATED/DEAD:

A thread is terminated due to any of these following reasons

- A thread dies naturally , when it completes its execution and exits run() method.
- A thread is forcefully stopped and killed by calling interrupt() method.

The thread is dead in this state, and no longer consumes any CPU cycle. Apart from this ,whenever we want to stop a thread from running further, we can call its **stop(**) method. This method causes the thread to move to the dead state. A thread will also move to the dead state automatically when it reaches the end of its method.
The **stop()** method may be used when the **premature death** of a thread is desired.

wait() and sleep() methods in java, are used to pause the execution of a particular thread in a multi threaded environment. The major differences between wait() and sleep() method are as follows.

Difference between Wait() and sleep() methods

Object.wait()	Thread.sleep()
wait() is an instance method, belonging to java.lang.Object class	sleep() method belongs to Thread class.
wait() can only be called from a synchronized block	*Thread.sleep()* is a static method that can be called from any context.
wait() method is always called on objects.	*sleep() method is always called on threads.*
It releases the lock on the object so that another thread can acquire the released object, by stopping current thread execution	*Thread.sleep()* **pauses the current thread for specified number of milliseconds.But it does not release any locks.**
In case of **wait()** method, thread goes in waiting state and it won't come back automatically until we call the notify() or notifyAll().	The sleep state can be terminated by calling the interrupt() method on thread object. And if the thread is interrupted, an **InterruptedException** is generated.
Wait() is used in concurrent access codes only.	sleep() is used whenever and wherever required.

4.1.7 Thread Priority:

Each thread has a priority associated with it in Java. In multithreading environment, thread scheduler schedulers them for priority based processes. Priorities can be set by JVM when it creates a thread or it can be given by programmer explicitly. Priorities a are represented by a number between 1 and 10. Java follows a preemptive scheduling. It means when a higher priority thread arrives for execution, the currently running lower priority thread is stopped.

There are 3 static variables defined in Thread class for priority.

public static int MIN_PRIORITY: This is minimum priority that a thread can have. Value for this is 1.

public static int NORM_PRIORITY: This is default priority of a thread if do not explicitly define it. Value for this is 5.

public static int MAX_PRIORITY: This is maximum priority of a thread. Value for this is 10.

By default, every thread is assigned NORM_PRIORITY i.e. 5. There are two methods defined in Thread class for assigning and retrieving priorities.

1. **public final int getPriority():** java.lang.Thread.getPriority() method returns priority of given thread.
2. **public final void setPriority(int newPriority):** java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.

Example 5: Program demonstrating use of getPriority() and setPriority() methods in multithreading.

```
class sum_thread extends Thread
{
  int sum=1;
  public void run()
{
        for(int i=1;i<=5;i++)
         {

          sum = sum + i;
          System.out.println("Sum = " +sum);
}
}
}
class mul_thread extends Thread
{
  int mul =1;
  public void run()
{
        for(int i=1;i<=5;i++)
         {
                mul = mul * i;
    System.out.println("multiplication = " + mul);
}
}
}
class priority_thread
{
  public static void main(String args[])
```

```
{
sum_thread st = new sum_thread();
Thread s_th = new Thread(st,"Sum Thread");

System.out.println("Default priority of sum thread is "+s_th.getPriority());
        s_th.setPriority(Thread.MIN_PRIORITY + 3);
 System.out.println("New priority of sum thread is "+s_th.getPriority());

mul_thread mt = new mul_thread();
Thread m_th = new Thread(mt,"Multiplication Thread");

  System.out.println("Default priority of Multiplication thread is
"+m_th.getPriority());
  m_th.setPriority(Thread.MAX_PRIORITY );
  System.out.println("New priority of Multiplication thread is "+m_th.getPriority());

 s_th.start();
m_th.start();

}
}
```

Output:

```
D:\JAVA>javac priority_thread.java

D:\JAVA>java priority_thread
Default priority of sum thread is 5
New priority of sum thread is 4
Default priority of Multiplication thread is 5
New priority of Multiplication thread is 10
multiplication = 1
Sum = 2
multiplication = 2
Sum = 4
multiplication = 6
Sum = 7
multiplication = 24
Sum = 11
multiplication = 120
Sum = 16

D:\JAVA>_
```

4.1.8 Multithreading using isAlive() and join():

Sometimes one thread needs to know when other thread is terminating. In java, isAlive() and join() are two different methods that are used to check whether a thread has finished its execution or not.

- **isAlive()** : The isAlive() method returns true if the thread upon which it is called is still running otherwise it returns false.
  General Syntax :
  final boolean isAlive()

- **join():**This method waits until the thread on which it is called terminates.
  It waits for the child thread to terminate and then joins the main thread.
  General Syntax :
  final void join() throws InterruptedException
  There are overloaded versions of join() method, which allows us to specify time for which you want to wait for the specified thread to terminate.
  final void join(long milliseconds) throws InterruptedException

Example 6 : Java program to create three threads. each thread should produce the sum of 1 to 10, 11 to 20and 21 to 30 respectively. Main thread should wait for all the threads to complete and it should
display the final sum.

```
class thread1 extends Thread
{
 int sum1 = 0,i;
 public void run()
{
for (i=1;i<=10;i++)
 {
  sum1 = sum1 + i;
 }
System.out.println("Thread1 sum " + sum1);
}
}

class thread2 extends Thread
{
 int sum2 = 0,i;
 public void run()
{
for (i=11;i<=20;i++)
```

```
{
 sum2 = sum2 + i;
 }
System.out.println("Thread2 sum " + sum2);
}
}
class thread3 extends Thread
{
 int sum3 = 0,i;
 public void run()
{
 for (i=21;i<=30;i++)
 {
sum3 = sum3 + i;
 }
System.out.println("Thread3 sum " + sum3);
}
}

class threethreads
{
  public static void main(String args[])
   {
    thread1 th1 = new thread1();
    thread2 th2 = new thread2();
    thread3 th3 = new thread3();

 Thread t1 = new Thread(th1);
    Thread t2 = new Thread(th2);
    Thread t3 = new Thread(th3);


t1.start();
   t2.start();
   t3.start();
System.out.println("Threads isAlive() status");
System.out.println("Thread 1 :" +t1.isAlive());
System.out.println("Thread 2 :"+t2.isAlive());
System.out.println("Thread 3 :"+t2.isAlive());

 try
 {
```

```
    t1.join();
    t2.join();
    t3.join();
   }
   catch (InterruptedException e)
   {
    System.out.println(e);
   }
  int total = th1.sum1 + th2.sum2 + th3.sum3;
  System.out.println("*****************************");
  System.out.println("The Final Total = " +total);
  System.out.println("*****************************");
  System.out.println("The Average= " +total/3);
  System.out.println("*****************************");

  System.out.println("Threads isAlive() status");
  System.out.println("Thread 1 :" +t1.isAlive());
  System.out.println("Thread 2 :"+t2.isAlive());
  System.out.println("Thread 3 :"+t2.isAlive());

  }
  }
```

Output:

```
D:\JAVA>javac threethreads.java

D:\JAVA>java threethreads
Threads isAlive() status
Thread 1 :true
Thread 2 :true
Thread 3 :true
Thread1 sum 55
Thread2 sum 155
Thread3 sum 255
*****************************
The Final Total = 465
*****************************
The Average= 155
*****************************
Threads isAlive() status
Thread 1 :false
Thread 2 :false
Thread 3 :false

D:\JAVA>_
```

4.2 APPLET:

4.2.1 Introduction:

We know that there are two types of Java programs Application and Applets. So far we have studied Applications, those are console based applications(CUI).

Applets are another special type of Java program. It is embedded with HTML code and run on the browser which is downloaded from the internet. It is the Graphical User Interface (GUI) part of java. To run these applets, the client should either have a Java enabled browser or a utility known as **appletviewer,** which comes as a part of JDK.

Applets contain numbers of graphical elements like mages,graphics,buttons,textboxes etc. It has the capability of displaying graphics, creating animations and playing sounds.

There are two ways of working with applets in Java.

- Applets evolved from "Applet" class. These applets use, the Abstract Window Toolkit (AWT) classes. Applets interact with the user through the AWT . An applet must be a subclass of the java.applet.Applet class.
- Applets based on Swing class, "JApplet".Swing provides a special subclass of the Applet class called javax.swing.JApplet. The JApplet class should be used for all applets that use Swing components to construct their graphical user interfaces (GUIs).

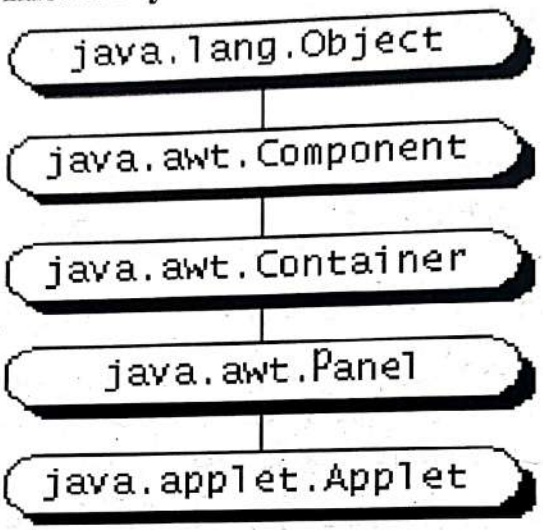
Differences between Application and Applets

Applet	Application
An Applet does not have a main() method.	An application always have a main() method to start its execution.
Applets are GUI based Java Program	Applications are CUI based Java Programs.
Applets cannot run independently, they have to be embedded inside a web page to get executed.	Applications can run independently.
Applets require a browser or an appletviewer to be executed.	Applications do not require any such utilities, they are executed at command line.
An applet cannot perform read and write operations on files stored on local disk.	An application can perform read and write operations on files stored on the local disk.
It requires high-security constraints as applets cannot be trusted.	It is a trusted application and doesn't require much security.

4.2.2 Applet Class:

An applet is created using the Applet class, which is a part of java.applet package. This Applet class provides an interface between the applet program and the Web Browser. It is responsible for executing applets in browser or appletviewer. Applet class provides several useful methods to give you full control over the execution of an applet.

Hierarchy of the Applet class



Few of the important methods of Applet class are listed below:

Methods of Applet class

Method	Description
public void init()	First method to be called when an applet begins execution
public void start()	Responsible for starting and resuming applet
public void stop()	Stops or suspends applet
public void destroy()	Terminates an applet
public Boolean isActive()	Returns true if applet is running, otherwise it retruns false.
public void resize(int width, int height)	Resizes the applet according to the specified width and height.
public void showStatus(String msg)	Displays the string msg, in the status bar of the browser or appletviewer.

4.2.3 Applet Structure:

Applet in java inherits its methods and properties from java.applet package well as from java.awt package. java.applet.Applet has methods like init(),start(),destroy() and

stop() which are responsible for creation and termination of an applet. But as we know, there is no main() method in java applets to start the execution of an applet. There is another method paint() in the container class, which is inherited by Applet class. This method is used to display output on screen in applet programs. The **paint( )** method is called each time an AWT-based applet's output must be redrawn. **paint( )** is also called when the applet begins execution. The **paint( )** method has one parameter of type Graphics. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

General Form : public void paint(Graphics g)

4.2.4 An Example Applet Program:

Here, is the first applet example

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

public class app1 extends Applet
{
        public void paint (Graphics g)
        {
        g.drawString("HELLO APPLET",100,130);
        }
}
```

Explanation :

1. The above java program begins with two import statements. The first import statement imports the Applet class from applet package. Every AWT-based(Abstract Window Toolkit) applet that you create must be a subclass (either directly or indirectly) of Applet class. The second statement imports the Graphics class from awt package.
2. The next line in the program declares the class app1. This class must be declared as public, because it will be accessed by code that is outside the program. Inside app1, **paint( )** is declared. This method is defined by the AWT and must be overridden by the applet.
3. Inside **paint( )** is a call to *drawString( )*, which is a member of the Graphics class. This method outputs a string beginning at the specified X,Y location. It has the following general form:

void drawString(String msg, int x, int y)

Here ,the msg "HELLO APPLET" will be displayed at (100,130) coordinates of the applet window.

Running the Applet:

There are **two** ways to run an applet :

1. Executing the applet within a Java-compatible web browser.
2. Using the standard tool like appletviewer. An applet viewer executes your applet in a window.

1. **Executing the applet within a Java-compatible web browser Remember** : Applet program should be saved as the same name as that of class name. Here, it is to be saved as **app1.java.**
To execute an applet in a web browser we have to write a short HTML text file that contains a tag that loads the applet. Type the following in your editor and save the file as app1.html

```
<HTML>
<BODY>
<applet code = app1 width = 300 height =350>
</applet>
</Body>
</HTML>
```

Again the 2 steps to run a java program

- javac app1.java
- app1.html (in Java enabled web browser)

2. **Using Appletviewer.**
To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer . Applet tag is always written in the comment, after importing the packages.

Example 1: Program to create an Applet using applet tag and appletviewer

```
/*
<applet code = app1 width = 300 height =350>

</applet>
*/

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

```
/*<applet code = app1 width = 300 height =350>
</applet>
*/

public class app1 extends Applet
{    public void paint (Graphics g)
        {       g.drawString("HELLO APPLET",100,130);  }
}
```

To run the above code:

- javac app1.java
- appletviewer app1.java

Output:

Explanation:

An applet window of widht and height ,300 and 350 respectively will be created. The String message "HELLO APPLET" will be displayed at (x,y) location of (100,130) specified in drawString() method.

4.2.5 Applet Life Cycle:

An applet in its life time moves from one state to another. There are various states of applet life cycle. They are:

- Born
- Running
- Idle
- Dead

Four methods in the Applet class give you the framework on which you build any applet. They are :

- init()
- start()
- stop()
- destroy()

Life cycle of An Applet:

Born State

When a new *applet* is born or created, it is activated by calling *init()* method. At this stage, new objects to the *applet* are created, initial values are set, images are loaded and the colors of the images are set. An *applet* is initialized only once in its lifetime, hence this method is called only once during the applets life time. Its general form is:

```
public void init()
{    //actions to be performed    }
```

Running State

An *applet* is in the running state when the system calls the *start()* method. This occurs as soon as the *applet* is initialized. An *applet* may also start when it is in idle state. At that time, the *start()* method is overridden. Its general form is:

```
public void start()
{  // actions to be performed }
```

Idle State

An *applet* comes in idle state when its execution has been stopped either *implicitly* or *explicitly*. An *applet* is *implicitly* stopped when we leave the page containing the currently running applet. An *applet* is *explicitly* stopped when we call *stop()* method to stop its execution. Its general form is:

```
public void stop()
{  // actions to be performed }
```

Dead State

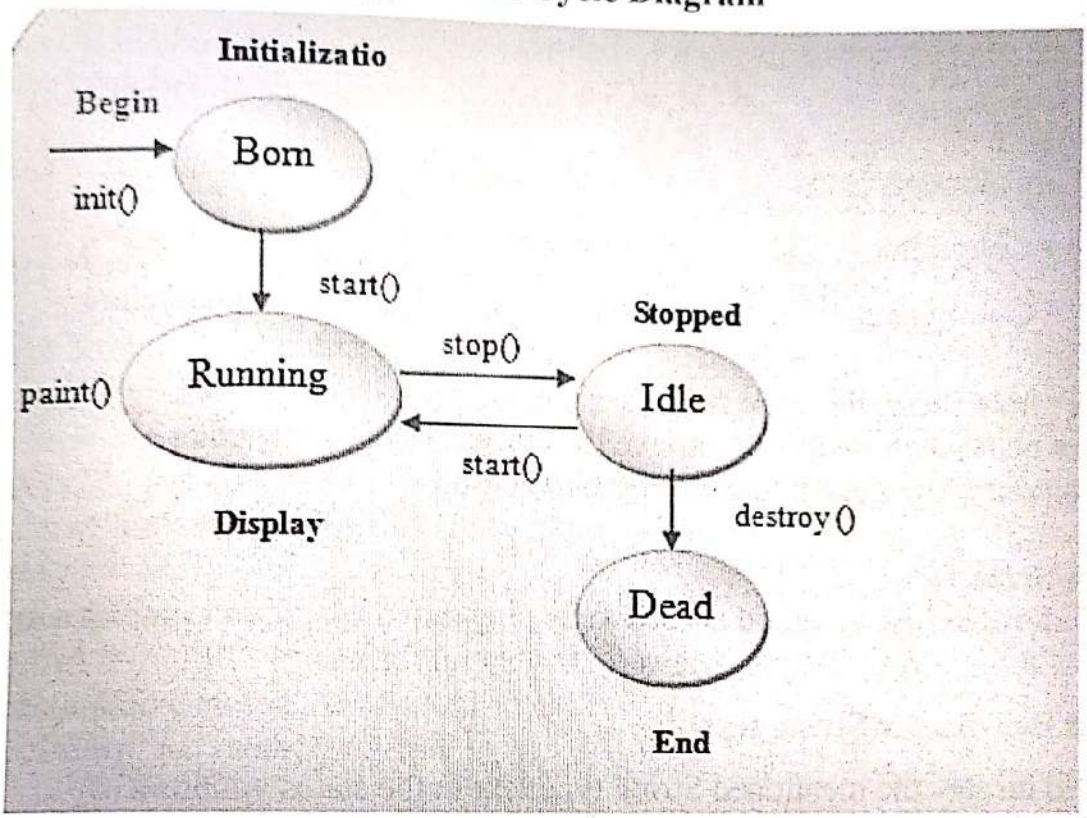
An *applet* is in dead state when it is removed from the memory. This can be done by calling destroy method(). Its general form is:

```
public void destroy
```

{ // actions to be performed }

Apart from the above stages, Java applet also possess *paint()* method. This method helps in drawing, writing and creating colored backgrounds of the applet. We have already discussed paint() method .

Applet Life Cycle Diagram



4.2.6 Common Methods Used In Displaying The Output:

Following are some common methods used in displaying the output.

- **drawstring()**

The drawstring() method, is a method of Graphics class, used to output a string to an applet. Its general form is :

void drawString(String str, int x, int y)

x and y are the coordinates where the string **str** is to be drawn on the frame.

Example :

```
public void paint(Graphics g)
{
        g.drawString("Hello",80,70)
}
```

- **setBackground()**

setBackground() method belongs to Component class. It is used to set the background color of the applet window. Its general form is :

void setBackground(Color mycolor)

➢ **setForeground()**

This method is used to set the foreground color i.e. the color of the text. Its general form is :

void setForeground(Color mycolor)

mycolor is one of the color constants or the new color created by the user

The list of color constants is given below:

- Color.red
- Color.orange
- Color.gray
- Color.darkGray
- Color.lightGray
- Color.cyan
- Color.pink
- Color.white
- Color.blue
- Color.green
- Color.black
- Color.yellow

Apart from these, there are two other methods :
Color getBackground() : to retrieve the current background color.
Color getForeground(): to get foreground color.

➢ **showStatus()**

The **showStatus( )** method is used to display the information in applet windows status bar of the browser or Appletviewer. Its general form is :

void showStatus(String text).

It will display the mentioned String type text in the status window.

Example 2: Program showing use of different Applet methods.

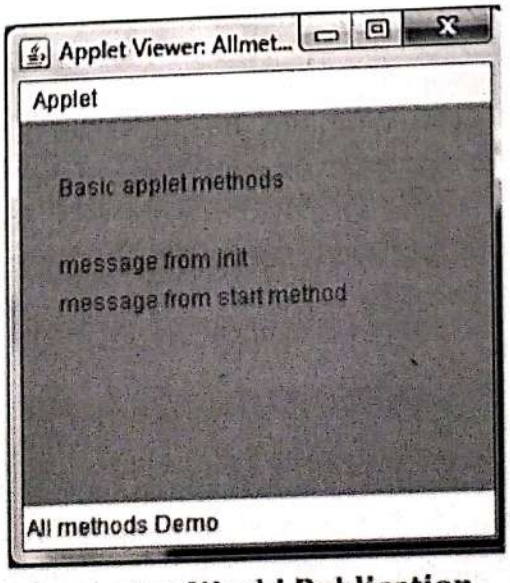
```
import java.applet.*;
import java.awt.*;
/*
<applet code = Allmethod width = 250 height =200>
</applet>
*/
public class Allmethod extends Applet
{
String mesg1, mesg2, mesg3, mesg4;
 public void init()
{
setBackground(Color.green);
 setForeground(Color.red);
mesg1 = "message from init";
}
public void start()
```

```
{
mesg2 = "message from start method";
}
public void stop()
{
mesg3 = "message from stop method";
}
public void destroy()
{
mesg4 = "system is destroying your applet";
}
public void paint(Graphics g)
{
g.drawString("Basic applet methods ",20,40);
if(mesg1 != null)
  g.drawString(mesg1,20,80);
if(mesg2 != null)
  g.drawString(mesg2,20,100);
if(mesg3 != null)
  g.drawString(mesg3,20,120);
if(mesg4 != null)
  g.drawString(mesg4,20,140);
showStatus("All methods Demo");
}
}
```

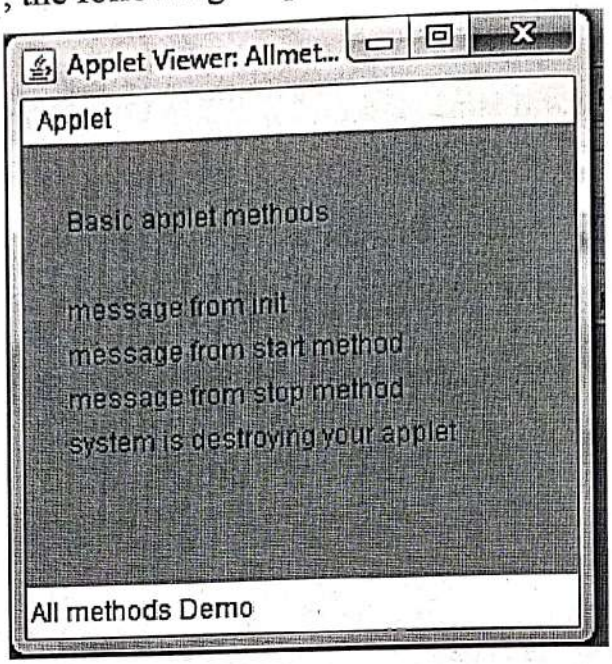
To run the above code:

- javac Allmethod.java
- appletviewer Allmethod.java

Output:



First the above output is generated. Now click on Applet menu, in the applet window and restart the applet. Then the messages from stop() and destroy() method will be displayed. Thereafter , the following output will be generated.



4.2.7 paint(), update() and repaint() methods:

- **paint()**

  This method is used to display output on screen in applet programs. The **paint(**) method is called each time an AWT-based applet's output must be redrawn. paint( ) is also called when the applet begins execution. The **paint( )** method has one parameter of type Graphics.

  General Form : public void paint(Graphics g)

- **update()**

  Whenever a screen needs redrawing, the **update()** method is called. By default, the **update()** method clears the screen and then calls the **paint()** method, which normally contains all the drawing code.

- **repaint()**

  The repaint() method is used to repaint the applet window. The repaint () method causes the AWT runtime system to execute the update () method of the Component class which clears the window with the background color of the applet and then calls the paint () method. Its calling sequence will be :

  repaint()--→update()-→paint()

4.2.8 More about Applet Tag:

HTML <applet> tag was used to embed the Java applet in an HTML document. Following is the full syntax of writing an applet tag. **Bold font** indicates something you should type in exactly as shown (except that letters don't need to be

uppercase). *Italic font* indicates that you must substitute a value for the word in italics. Square brackets ([and]) indicate that the contents of the brackets are optional.

< APPLET
  [CODEBASE = *codebaseURL*]
  CODE = *appletFile*
  [ALT = *alternateText*]
  [NAME = *appletInstanceName*]
  WIDTH = *pixels*
  HEIGHT = *pixels*
  [ALIGN = *alignment*]
  [VSPACE = *pixels*]
  [HSPACE = *pixels*]
>
[< PARAM NAME = *appletParameter1* **VALUE** = *value* **>]**
[< PARAM NAME = *appletParameter2* **VALUE** = *value* **>]**

. . .
[*alternateHTML*]
</APPLET>

CODEBASE = *codebaseURL*
This optional attribute specifies the base URL of the applet -- the directory or folder that contains the applet's code. If this attribute is not specified, then the document's URL is used.

CODE = *appletFile*
This required attribute gives the name of the file that contains the applet's compiled Applet subclass. This file is relative to the base URL of the applet. It cannot be absolute.

ALT = *alternateText*
This optional attribute specifies any text that should be displayed if the browser understands the APPLET tag but can't run Java applets.

NAME = *appletInstanceName*
This optional attribute specifies a name for the applet instance, which makes it possible for applets on the same page to find (and communicate with) each other.

WIDTH = *pixels*

HEIGHT = *pixels*
These required attributes give the initial width and height (in pixels) of the applet display area, not counting any windows or dialogs that the applet brings up.

ALIGN = *alignment*

This required attribute specifies the alignment of the applet. The possible values of this attribute are the same (and have the same effects) as those for the IMG tag: left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom.

VSPACE = *pixels*

HSPACE = *pixels*

These optional attributes specify the number of pixels above and below the applet (VSPACE) and on each side of the applet (HSPACE). They're treated the same way as the IMG tag's VSPACE and HSPACE attributes.

< PARAM NAME = *appletParameter1* **VALUE** = *value* >

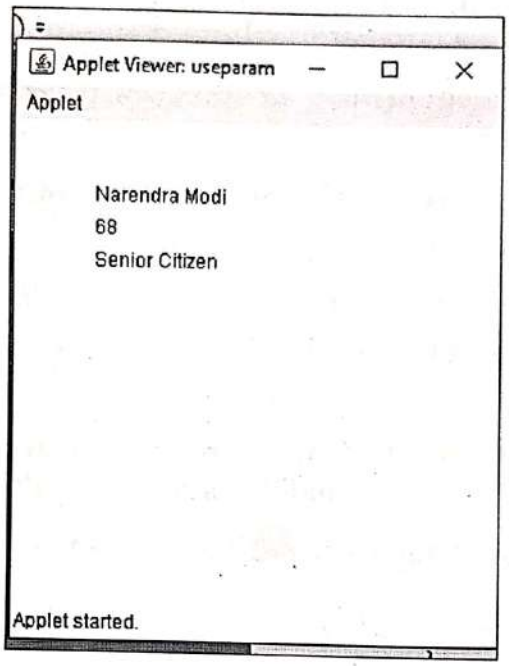
<PARAM> tags are the only way to specify applet-specific parameters. Applets read user-specified values for parameters with the getParameter() method.

General form : String getParameter(String name)

Example 3 : Program showing use of Param tag

```
import java.applet.*;
import java.awt.*;
/* <applet code="useparam" width="300" height="300">
<param name="Name" value="Narendra Modi">
<param name="Age" value="68">
</applet> */
public class useparam extends Applet
{  int age;
   public void paint(Graphics g)
       {
          String str_name=getParameter("Name");
          String str_age=getParameter("Age");
          g.drawString(str_name,50, 50);
         g.drawString(str_age,50,70);
// as str_age is a string variable we have to convert it to int, to perform any
operations on it
       age = Integer.parseInt(str_age);
       if(age>60)
                g.drawString("Senior Citizen",50,90);
       else
                g.drawString("Not a Senior Citizen",50,90);
   }
}
```

Output:



4.2.9 Methods of Graphics class:

The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well. It belongs to java.awt package, defined as
public abstract class Graphics extends Object

There are many methods defined in Graphics class. They are as under:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.

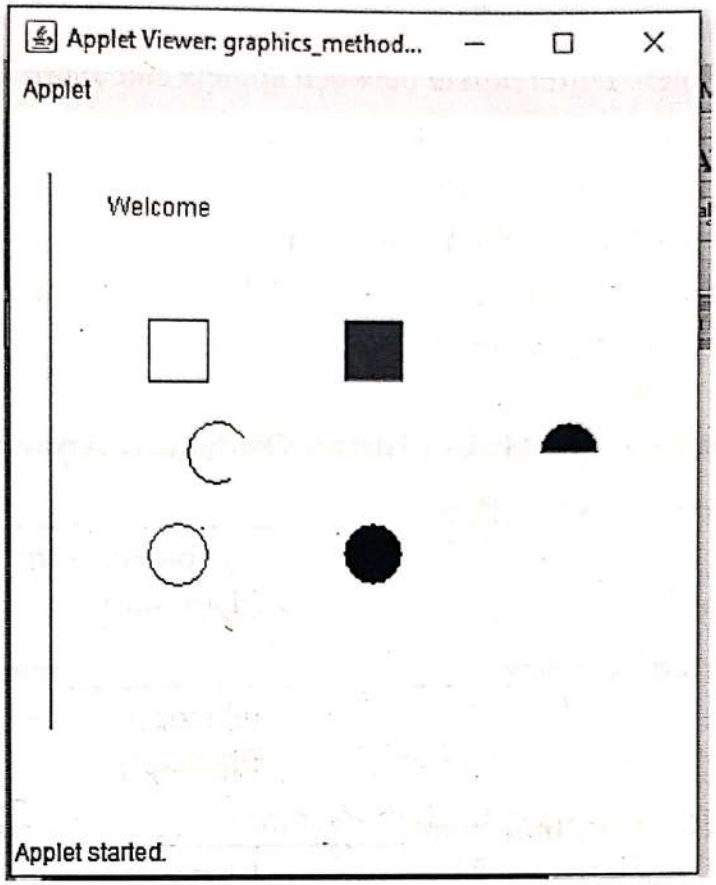
8. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
9. **abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints):** Draws a closed polygon defined by arrays of x and y coordinates.
10. **abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints):** Fills a closed polygon defined by arrays of x and y coordinates.
11. **abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight):** Draws an outlined round-cornered rectangle using this graphics context's current color.
12. **abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight):** Fills the specified rounded corner rectangle with the current color.

Example 4: Program showing use of different methods of Graphics class

```
import java.applet.*;
import java.awt.*;
 /*
<applet code="graphics_methods.class" width="350" height="350">
</applet> */
public class graphics_methods extends Applet
{
  public void paint(Graphics g)
  {
    g.setColor(Color.red);
    g.drawString("Welcome",50, 50);
    g.drawLine(20,30,20,300);
    g.drawRect(70,100,30,30);
    g.fillRect(170,100,30,30);
    g.drawOval(70,200,30,30);

    g.setColor(Color.blue);
    g.fillOval(170,200,30,30);
    g.drawArc(90,150,30,30,30,270);
    g.fillArc(270,150,30,30,0,180);
  }
}
```

Output



Exercises

Answer the following question.

1. What is a Thread? Discuss : Multithreading in Java
2. Write a note on : java.lang.Thread class.
3. State the differences between Process Based multitasking and Thread based multitasking.
4. Explain giving example, different ways of creating threads in java.
5. State and explain the different states of Thread in java
6. What is Thread Priority? How can we set and retrieve priority for a thread?

7. Differentiate between : wait() and sleep()
8. Explain : isAlive() and join()
9. What is an applet? Differentiate between applets and application.
10. Write a detailed note on Applet class
11. Describe the life cycle of an applet.
12. Explain : paint(),update() and repaint() methods
13. Explain the different parameters of APPLET Tag
14. State and explain different methods of Graphics class.

Fill in the Blanks / Multiple choice Question-Answer.

1. Java achieved multitasking using _____.
   (a) class
   (b) inheritance
   (c) Polymorpism
   (d)Thread
2. A thread is started by calling _____ method .
   (a) start()
   (b) run()
   (c) stop()
   (d)sleep()
3. A thread starts its execution from _____ method.
   (a) start()
   (b) run()
   (c) stop()
   (d) sleep()
4. _____ method suspends a thread for specified time.
   (a) start()
   (b) run()
   (c) stop()
   (d) sleep()
5. _____method returns a reference to the thread on which it is called.
   (a) isAlive()
   (b) join()
   (c) currentThread()
   (d)None of above
6. By default ,_____ package is implemented in every Java program.
   (a) java.util
   (b) java.net
   (c) java.lang
   (d) java.awt()
7. A thread lies is in _____ state when it calls a method with a time out parameter.
   (a) new
   (b) blocked
   (c) terminated
   (d) timed waiting
8. _____method waits until the thread on which it is called terminates.
   (a) join()
   (b) stop()
   (c) interrupt()
   (d) run()

9. Which tool is used to compile java applet? _____
   (a) java
   (b) appletviewer
   (c) javac
   (d)none of the above

10. Which tool is used to execute an applet? _____
   (a) java
   (b) appleviewer
   (c) javac
   (d) none of the above

11. Which of the following is not a method of Applet class? _____
   (a) start()
   (b) init()
   (c) stop()
   (d) all of above

12. _____ method does the same job as the paint method.
   (a) update
   (b) stop
   (c) destroy
   (d)start

13. The _____ method is called each time an AWT-based applet's output must be redrawn.
   (a) paint
   (b) repaint
   (c) update
   (d)All of the above

14. _____ method is used to read parameters from an applet.
   (a) getCodeBase()
   (b) getClassName()
   (c) getDocumentBase()
   (d) getParamenter()

15. _____ method is used to draw a rectangle.
   (a) drawRect
   (b) fillRect
   (c) drawRoundRect
   (d) fillRoundRect

16. From which package does Applet inherits its properties and methods ? _____
   (a) java.lang
   (b) java.util
   (c) java.applet
   (d) None of the above

Answers:

1 – d	2 – a	3 – b	4 – d
5 – c	6 – c	7 – d	8 – a
9 – c	10 – b	11 – d	12 – a
13 – a	14 – d	15 – a	16 – c

CC-214 Core Java Practical (UNIT – 4)

Program : 1

Write an application that starts two threads. First thread displays even numbers in the range specified from the command line and second thread displays odd numbers in the same range. Each thread waits for 300 milliseconds before displaying the next numbers. The application waits for both the thread to finish and then displays the message "Both threads completed".

```
public class Prog1_oddeventhread {
  public static void main(String[] args)
  {
    String s = args[0];
    Runnable r = new oddthread(s);
    Thread t = new Thread(r);
    Runnable r2 = new eventhread(s);
    Thread t2 = new Thread(r2);
      t.start();
    t2.start();
      try
      {
            t.join();
            t2.join();
      }
       catch (InterruptedException e)
      {
            System.out.println(e);
      }
      System.out.println("End of Main Thread : Both threads completed");
      }
      }
class oddthread implements Runnable
{
 int n;
oddthread(String s)
 { n = Integer.parseInt(s);}
 public void run(){
   for(int i=0;i<n;i++){
     try
```

```
        {
        if(i%2 == 1)
          {
System.out.println("odd "+i);
Thread.sleep(300);

          }
        } catch(InterruptedException e)
       {  }
     }
   }
}
class eventhread implements Runnable
{
int n;
eventhread(String s)
  { n = Integer.parseInt(s);}

 public void run(){
     for(int i=0;i<n;i++){
       try{
        if(i%2 == 0)
          { System.out.println("even "+i);
Thread.sleep(300);
          }
        } catch (InterruptedException e)
        { }
     }
   }
}
```

Output:

```
D:\java_unit4>javac Prog1_oddeventhread.java

D:\java_unit4>java Prog1_oddeventhread 8
odd 1
even 0
odd 3
even 2
odd 5
even 4
odd 7
even 6
End of Main Thread : Both threads completed
```

Program : 2

Write a program that create and starts five threads. Each thread is instantiated from the same class. It executes a loop with ten iterations. Each iteration displays the character 'x' and sleep for 500 milliseconds. The application waits for all threads to complete and then display a message 'hello'.

```
class thread1 extends Thread
{
 int i;
 public void run()
 {
  for (i=1;i<=10;i++)
  {
   try
        {
                System.out.println("T1---x" + i);
                Thread.sleep(1000);
    } catch(InterruptedException e)
   {  }
  }
 }
}
class thread2 extends Thread
{
 int i;
 public void run()
 {
  for (i=1;i<=10;i++)
  {
   try
   {
        System.out.println("T2---x" + i);
        Thread.sleep(1000);
    } catch(InterruptedException e)
   {  }

  }
 }
}
```

```
class thread3 extends Thread
{
 int i;
 public void run()
{
  for (i=1;i<=10;i++)
   {
   try{

System.out.println("T3---x" + i);
Thread.sleep(1000);
    } catch(InterruptedException e)
   {  }

   }
 }
}
 class thread4 extends Thread
 {
  int i;
  public void run()
 {
   for (i=1;i<=10;i++)
   {
   try{
         System.out.println("T4---x" + i);
         Thread.sleep(1000);
    } catch(InterruptedException e)
   {  }

   }
 }
}

class Prog2_fivethreads
{
  public static void main(String args[])
   {
    thread1 th1 = new thread1();
    thread2 th2 = new thread2();
    thread3 th3 = new thread3();
```

```
thread4 th4 = new thread4();
Thread t1 = new Thread(th1);
Thread t2 = new Thread(th2);
Thread t3 = new Thread(th3);
Thread t4 = new Thread(th4);

t1.start();
t2.start();
t3.start();
t4.start();

try
{
 t1.join();
 t2.join();
 t3.join();
 t4.join();
}
 catch (InterruptedException e)
{
System.out.println(e);
}

System.out.println("**********HELLO**********");
}
}
```

Output:

```
D:\java_unit4>javac Prog2_fivethreads.java

D:\java_unit4>java Prog2_fivethreads
T1---x1
T2---x1
T3---x1
T4---x1
T1---x2
T2---x2
T1---x3
T2---x3
T3---x2
T4---x2
T1---x4
T2---x4
T1---x5
T3---x3
T2---x5
T4---x3
T1---x6
T2---x6
T1---x7
T2---x7
T3---x4
T4---x4
T1---x8
T2---x8
T1---x9
T2---x9
T3---x5
T4---x5
T1---x10
T2---x10
T3---x6
T4---x6
T3---x7
T4---x7
T3---x8
T4---x8
T3---x9
T4---x9
T3---x10
T4---x10
**********HELLO**********
```

Program 3:

Write a java program to create 2 threads each thread calculates the sum and average of 1 to 10 and 11 to 20 respectively. After all thread finish, main thread should print message " Task Completed". Write this program with use of runnable interface.

```
class thread1 implements Runnable
{
int sum1 = 0,i;
public void run()
{
 for (i=1;i<=10;i++)
  {
   try
   {
   sum1 = sum1 + i;
   Thread.sleep(1000);
```

```
  }
 catch (InterruptedException e)
   { }
 }
System.out.println("Thread1 sum " + sum1);
System.out.println("Averge of 1 to 10 numbers is "+ (sum1/10));
}
}

class thread2 implements Runnable
{
 int sum2 = 0,i;
 public void run()
{
 for (i=11;i<=20;i++)
 {
 try
 {
  sum2 = sum2 + i;
  Thread.sleep(2000);
  }
 catch (InterruptedException e)
  { }
 }
System.out.println("Thread2 sum " + sum2);
System.out.println("Averge of 11 to 20 numbers is "+ (sum2/10));
  }
}

class Prog3_twosumthreads
{
  public static void main(String args[])
  {
    thread1 th1 = new thread1();
    thread2 th2 = new thread2();
    Thread t1 = new Thread(th1);
    Thread t2 = new Thread(th2);
    t1.start();
    t2.start();
  try
  {
```

```
    t1.join();
    t2.join();

   }
   catch (InterruptedException e)
  {
    System.out.println(e);
  }
  int total = th1.sum1 + th2.sum2 ;
  System.out.println("*****************************");
  System.out.println("The Final Total = " +total);
  System.out.println("*****************************");
  System.out.println("The Average = " +total/2);
  System.out.println("*********TASK COMPLETED*********");
   }
  }
```

Output:

```
D:\java_unit4>javac Prog3_twosumthreads.java

D:\java_unit4>java Prog3_twosumthreads
Thread1 sum 55
Averge of 1 to 10 numbers is 5
Thread2 sum 155
Averge of 11 to 20 numbers is 15
*****************************
The Final Total = 210
*****************************
The Average = 105
*********TASK COMPLETED**********
```

Program 4:

Create two thread. One thread print 'fybca' 4 times and another thread print 'sybca' 6 times. Set priority for both thread and when thread finished print 'tybca' from main.

```
class fythread extends Thread
{
  public void run()
{
  for(int i=1;i<=4;i++)
  {
   try
   {
```

```
            System.out.println("FYBCA ");
            Thread.sleep(1000);
        }
        catch(InterruptedException e)
        {}
    }
}
}
class sythread extends Thread
{
  public void run()
{
  for(int i=1;i<=6;i++)
  {
            System.out.println("SYBCA ");
    }
  }
}
class prog4_bcathreads
{
  public static void main(String args[])
  {
            fythreadfyt = new fythread();
            Thread fyth = new Thread(fyt,"First Year Thread");
            System.out.println("Name of the thread = "+fyth.currentThread());
            fyth.setPriority(Thread.MAX_PRIORITY - 3);
            sythreadsyt = new sythread();
            Thread syth = new Thread(syt,"Second Year Thread");
            System.out.println("Name of the thread = " + syth.currentThread());
            syth.setPriority(Thread.MIN_PRIORITY + 3 );
            fyth.start();
            syth.start();
             try
             {
                    fyth.join();
                    syth.join();
            } catch (InterruptedException e)
            { }
System.out.println("TYBCA");
  }
}
```

Output:

```
D:\java_unit4>javac prog4_bcathreads.java

D:\java_unit4>java prog4_bcathreads
Name of the thread : Thread[main,5,main]
Name of the thread : Thread[main,5,main]
FYBCA
SYBCA
SYBCA
SYBCA
SYBCA
SYBCA
SYBCA
FYBCA
FYBCA
FYBCA
TYBCA
```

Program 5:

Create an applet which draws a line, rectangle and filled circle in applet display area.

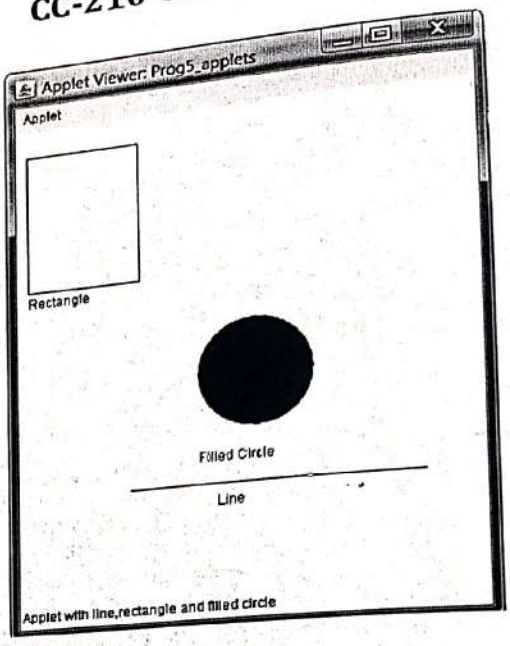
```
import java.applet.*;
import java.awt.*;

/* <applet code = Prog5_applets width = 400 height = 450>
   </applet>   */

public class Prog5_applets extends Applet
{

 public void paint(Graphics g)
{
 int x=10,y = 25;
int width = 100,height = 130;
g.drawRect(x,y,width,height);
g.drawString("Rectangle",10,170);
g.drawLine(100,350,350,350);
g.drawString("Line",175,370);
g.fillOval(160,200,100,100);
g.drawString("Filled Circle",160,330);
showStatus("Applet with line,rectangle and filled circle");
}
}
```

Output:



Program 6:

Write applets to draw the following shapes.

a. cone
b. cylinder
c. cube

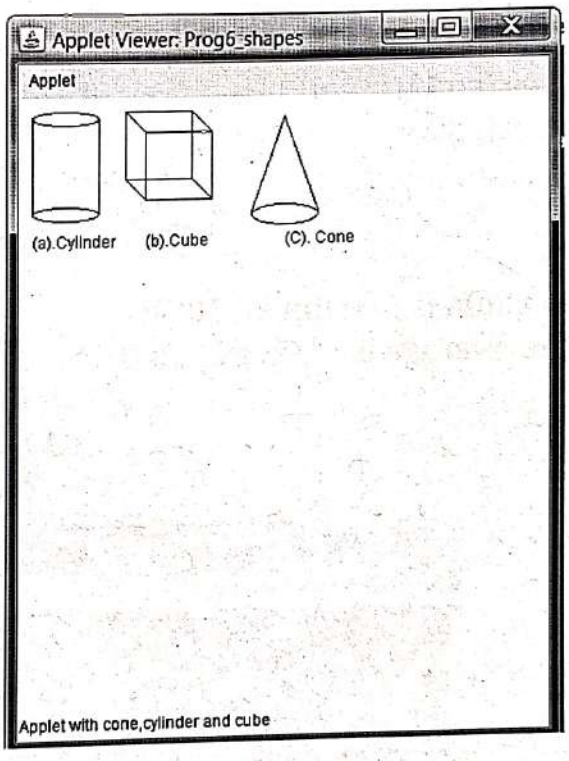
```
import java.applet.*;
import java.awt.*;

/*
<applet code = Prog6_shapes width = 400 height = 450>
</applet> */

public class Prog6_shapes extends Applet
{
 public void paint(Graphics g)
{
/*Cylinder*/
g.drawString("(a).Cylinder",10,110);
g.drawOval(10,10,50,10);
g.drawOval(10,80,50,10);
g.drawLine(10,15,10,85);
g.drawLine(60,15,60,85);
/*Cube*/
g.drawString("(b).Cube",95,110);
g.drawRect(80,10,50,50);
g.drawRect(95,25,50,50);
```

```
g.drawLine(80,10,95,25);
g.drawLine(130,10,145,25);
g.drawLine(80,60,95,75);
g.drawLine(130,60,145,75);
/*Cone*/
g.drawString("(C). Cone", 200,110);
g.drawLine(200,15,175,85);
g.drawLine(200,15,225,85);
g.drawOval(175,80,50,15);
showStatus("Applet with cone,cylinder and cube");
 }
}
```

Output:



Program 7:

Write an applet that take 2 numbers as parameter and display their average and sum.

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

/*<applet code = Prog7_sumavg width = 400 height = 400>
<param Name = "a1" Value = "10">
<param Name = "a2" Value = "20">
</applet>*/
```

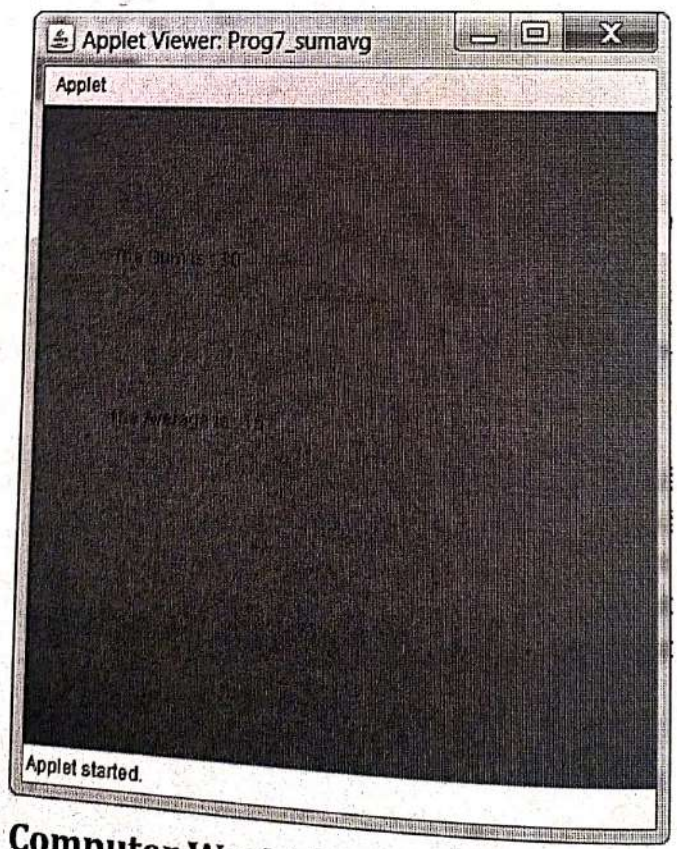
```
public class Prog7_sumavg extends Applet
{
public void init()
{
setBackground(Color.red);
setForeground(Color.blue);
}
public void paint(Graphics g)
{
String str1,str2;
int n1,n2,sum=0,avg=0;

str1=getParameter("a1");
str2=getParameter("a2");

n1=Integer.parseInt(str1);
n2=Integer.parseInt(str2);

sum=n1+n2;
avg=sum/2;
g.drawString("The Sum is : "+sum,50,100);
g.drawString("The Average is : "+avg,50,200);
}
}
```

Output:



Program 8:

Write a Java applet that draws a circle centred in the centre of the applet. The radius of the circle should be passed as a parameter

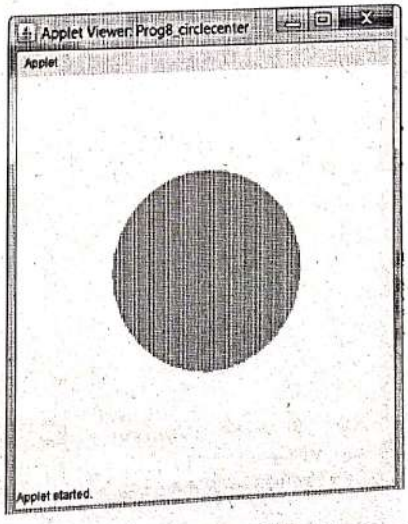
```
import java.applet.*;
import java.awt.*;
import java.util.Random;
/*
<applet code = Prog8_circlecenter width = 400 height = 400>
<param Name = "radius" Value = "100">
</applet> */

public class Prog8_circlecenter extends Applet
{

public void paint(Graphics g)
{
  Dimension d = getSize();
  int xc = d.width/2;
  int yc=d.width/2;
  String str1;
  int r1;
  str1=getParameter("radius");
  r1=Integer.parseInt(str1);
  g.setColor(Color.cyan);
  g.fillOval(xc-r1,yc-r1,2*r1,2*r1);
}
}
```

Note: To draw circle also, drawOval() or fillOval() method has to be used.

Output:



Program 9:

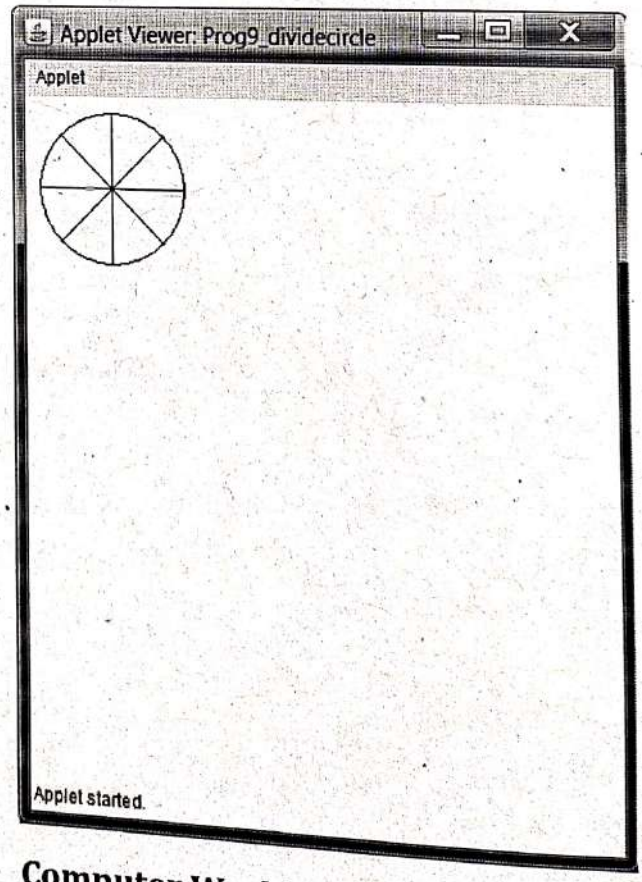
Write an applet that draw a circle divided in 6 equal parts.

```
import java.awt.*;
import java.applet.*;
/*
<applet code = Prog9_dividecircle width = 400 height = 450>
</applet> */

public class Prog9_dividecircle extends Applet
{

public void paint(Graphics g)
{
g.drawOval(10,10,100,100);
g.drawLine(60,10,60,110);
g.drawLine(10,60,110,60);
g.drawLine(25,25,95,95);
g.drawLine(95,25,25,95);
}
}
```

Output:



Program 10:

Write an applet that draw a rectangle divided in 5 equal parts.

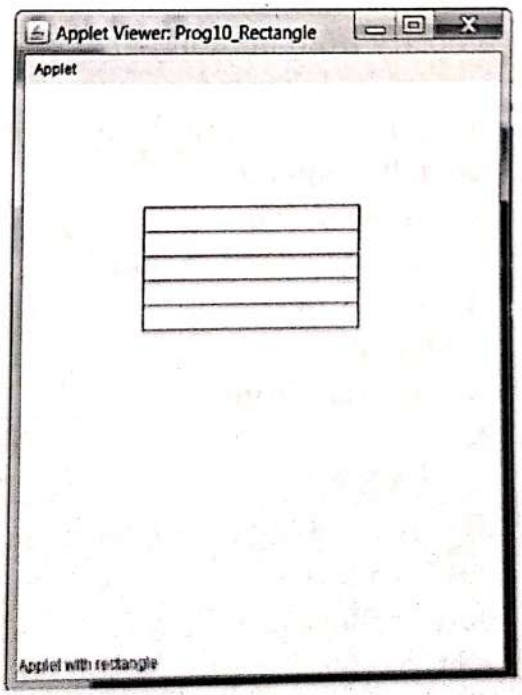
```
import java.applet.*;
import java.awt.*;
/*
<applet code = Prog10_Rectangle width = 400 height = 450>
</applet> */

public class Prog10_Rectangle extends Applet
{

 public void paint(Graphics g)
{
g.drawRect(100,100,180,100);
g.drawLine(100,120,280,120);
g.drawLine(100,140,280,140);
g.drawLine(100,160,280,160);
g.drawLine(100,180,280,180);

showStatus("Applet with rectangle");
}
}
```

Output:



AA-101

April-2019

B.C.A., Sem-IV

CC-210 : Core Java

[Max. Marks : 70

Time : 2:30 Hours]

1. **(A) Write the following:**

(i) Explain the priciples of Object-Oriented Programming Languages. 7

(ii) Write a short note on:

(A) Types of Operators 4

(B) Method Overloading 3

OR

(i) Which Java features replace C++in most of the application development? 7

(ii) Write a short note on:

(A) Importance of JVM. 4

(B) Constructor and its types. 3

(B) Do as directed. (any **four** out of **six)** 4

(i) Which of the following is not an object-oriented language?

(A) Simula (B) Java

(C) C (D) javadoc

(ii) Which command is used to interpret a Java program file?

(A) Java (B) javac

(C) Javap (D) javadoc

(iii) Which property from the following is correct for constructors?

(A) can not be inherited

(B) an <int> method implicitly created

(C) Default is inbuilt

(D) All of the above

(iv) Which of these statement is valid?

(A) short s1 = 143; (B) int jl = '3;'

(C) double dl = 6.3; (D) float fl = 4.3;

(v) To perform bitwise Oring, _____ operator is used in Java.

(A) & (B) !

(C) XXXXXX (D) XXXXX

(vi) Which of the following is true for class product?

(A) new product()

(B) new product(product s)

(C) new product(int pno, String pname)

(D) All of the above

2. (A) Write the following :

(i) "Multiple values can be passed to and returned from method using array." Explain with suitable examples. 7

(ii) Write a short note on:

(A) Interface extending 4

(B) Method Overriding 3

OR

(i) What is Interface? What are its types? 7

(ii) Write a short note on:

(A) Difference between Abstract class and interface 4

(B) Command-line arguments 3

(B) Do as directed. (any four out of six) 4

(i) Which statements from the following is correct?

(A) int m[ ] = new int [5] [5];

(B) int [ ] m [ ] = new int [5] [5];

(C) int [ ] [ ] = new int [5] [5];

(D) None of the above

(ii) _____ varables methods are allowed in static method.

(A) only static (B) only non-static

(C) static and non-static both (D) static, non static, final.

Abstract

(iii) Which of the following keyword allows deriving members from anothe class?

(A) extends (B) implements

(C) derives (D) inherits

(iv) Child class method hides execution of parent class method is an example of _____

(A) overloading (B) loadingover

(C) overbiding (D) overriding

3. (A) Write the following:

(i) Draw Access Protection chart. Explain its use inside and outside package with example. 7

(ii) What is multi-catch in exception handling? Explain with example 7

OR

(i) What is wrapper class? Explain any two with suitable examples. 7

(ii) Explain user defined exception with example. 7

(B) Do as Directed. (any three out of five) 3

(i) _____ must be included at the top of every java source file.

(A) import (B) impact
(C) inherit (D) package

(ii) What is the recult of complling the following of code: package pkg1 .pkg2;
(A) Compile time error (B) runtime error
(C) Compile but no execution (D) compile and execution

(iii) _____ is the immediate parent class of all exception in Java.
(A) Exception (B) throws
(C) Throw (D) Catch

(iv) _____ is always specifide after method that does nit catch the exception it throws.
(A) throw (B) throws
(C) finally (D) new

(v) Which of the following statement will not generate exception?
(A) z = x/0; (B) int a[] = new int[5];a[5]=5;
(C) a=10; try { a=20;} (D) None of the above

4. (A) Write the following :

(i) Which are the two ways of creating threads? Explain with examples. 7

(ii) List methods of Graphics class in applet and explain any two with an example. 7

OR

(i) How Java provides multithreding concept to reduce overall execution time? 7

(ii) Draw Applet Life cycle and explain with suitable example. 7

(B) Do as Directed. (any three out of five) 3

(i) Which package contains all methods and variables of multithreding?
(A) java.thread (B) java.lang
(C) java.Object (D) Java.package

(ii) Once a thread is created, it enters into _____ state.
(A) Ready (B) Running
(C) New (D) Run

(iii) _____ has same values for starting and ending co-ordinates.
(A) Polygon (B) Polyline
(C) Polyrectangle (D) Polycircle

(iv) _____ is the method of an applet.
(A) init( ) (B) display( )
(C) execute ( ) (D) None of the above

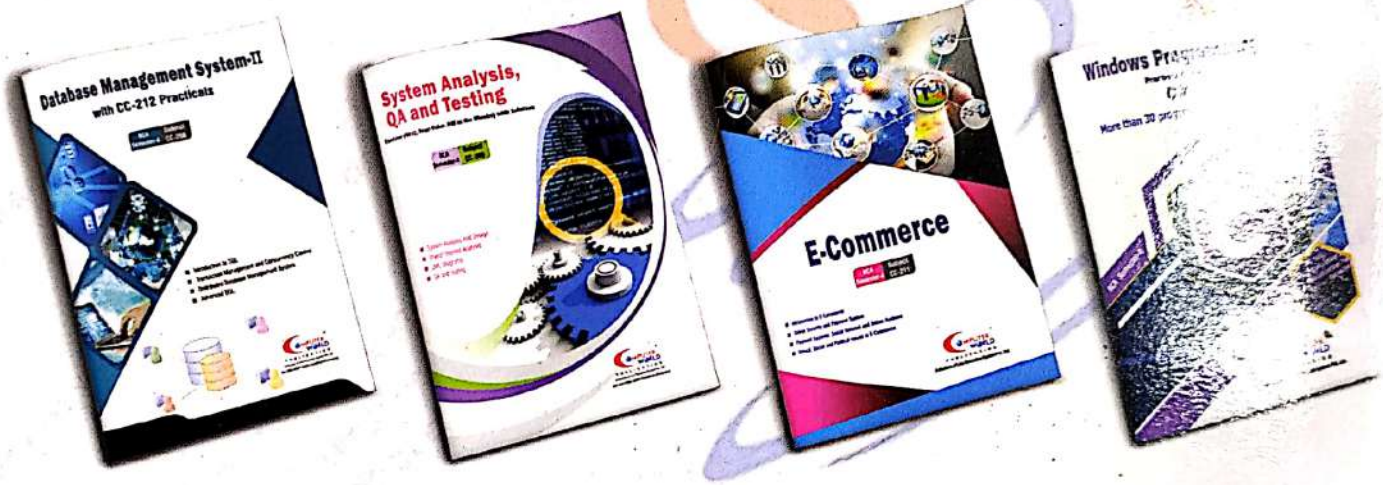
(v) _____ class has methods to draw graphical symbols on the applet.
(A) Drawing (B) Graph
(C) Graphics (D) Image

☺ ☺ ☺ ☺ ☺

According to the New Syllabus of the Gujarat University

Study Material for Bachelor of Computer Application (BCA)

Exercise (MCQ, True-False, Fill in the Blanks) with Solution



More Books for BCA

Semester-1, Semester-2, Semester-3, Semester-4, Semester-5 and Semester-6

Shop Online @ www.computerworld.ind.in

COMPUTER WORLD PUBLICATION
A Division of Live Education System Pvt. Ltd.
An ISO 9001:2008 Certified Company

COMPUTER WORLD PUBLICATION
43, 5th Floor, Sanidhya Complex, Opp. Sanyas Ashram,
Nr. M. J. Library,Ashram Road, Ahmedabad-380009.
Ph. : (079) 26580723 / 823, 97240 11150, 97250 22917
URL: www.computerworld.ind.in, www.cworld.co.in
e-Mail : info@computerworld.ind.in

978-93-88092-24-1
Book Code : CEBCA116 | Price ₹ 185/-