



# V2V EDTECH LLP

Online Coaching at an Affordable Price.

## OUR SERVICES:

- Diploma in All Branches, All Subjects
- Degree in All Branches, All Subjects
- BSCIT / CS
- Professional Courses



**+91 93260 50669**



**v2vedtech.com**



**V2V EdTech LLP**



**v2vedtech**



---

## Unit-I Basic Syntactical constructs in java

### Introduction to java

- James Gosling of sun micro-system created java in 1991.
- The original name of java is OAK. OAK is a name of tree, but later the language was change in the JAVA.
- Belongs to Sun- stand ford university network
- In 2010 jan 27 oracle corporation acquired sun microsystem.

#### Java Application/packages(edition)(API) (in 1995,completed java with three edition)

- By using java, we can develop the desktop application, enterprise edition application, & device application.
- To develop desktop application JSE is used(java platform standard edition)
- To develop enterprise application JEE is used(java platform enterprise edition)
- To develop device application JME is used.(java platform micro-edition).
- Old version are J2SE, J2ME, J2EE &
- new version are JSE, JME, JEE
- Core java belongs to JSE.

### 1.1. Java Features and Java Programming Environment

#### 1.1.1. Java Features

There are many features of JAVA. They are also known as JAVA buzzwords.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed



---

## 1. Simple

- JAVA is Easy to write and more readable and eye catching
- JAVA has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drawn from C++ thus making JAVA learning simpler

## 2. Object-Oriented

- JAVA programming is object-oriented programming language
- Like C++, JAVA provides most of the object oriented features
- JAVA is pure OOPS Language. (while C++ is semi object oriented).

## 3. Platform independent

- A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. JAVA provides software based platform.
- The JAVA platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.

It has two components.

- Runtime Environment
- API(Application Programming Interface)

JAVA code can be run on multiple platforms E.G.–Windows, Linux, Sun Solaris, Mac/OS etc. JAVA code is compiled by the compiler and converted into bytecode.

This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

## 4. Secured

JAVA is secured because

- No explicit pointer
- Programs run inside virtual machine sandbox.
- **Class loader** – adds security by separating the package for the classes of the local file system from those that are imported from network sources
- **Byte code Verifier** – checks the code fragments for illegal code that can violate access right to objects
- **Security Manager** – determines what resources a class can access such as reading and writing to the local disk These security are provided by JAVA language. Some security can also be provided by application developer through SSL, JAAS, cryptography etc.



---

## 5. Robust

- JAVA makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which JAVA improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.

## 6. Architecture neutral

- JAVA is not tied to a specific machine or operating system architecture
- Machine Independent i.e JAVA is independent of hardware
- Compiler generates byte codes, which have nothing to do with a particular computer architecture, hence a JAVA program is easy to interpret on any machine

## 7. Portable

- JAVA programs can execute in any environment for which there is a JAVA runtime system(JVM)
- JAVA programs can be run on any platform (Linux, Window, Mac)
- JAVA programs can be transferred over world wide web (e.g applets)

## 8. Dynamic

- JAVA programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time

## 9 Interpreted

- JAVA supports cross-platform code through the use of JAVA bytecode
- Byte code can be interpreted on any platform by JVM

## 10. High Performance

JAVA is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, JAVA enables high performance with the use of just-in-time compiler.

## 11. Multithreaded

A thread is like a separate program, executing concurrently. One can write JAVA programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

## 12. Distributed

- JAVA was designed with the distributed environment



- JAVA can be transmit, run over internet

### 1.1.2. Java Programming Environment

- JAVA development environment includes a number of development tools, classes and methods.
- It is a part of the system known as JAVA Development kit (JDK).
- The classes and methods are part of the JAVA standard library known as JSL, and it is known as the application Programming Interface(API).
- The java runtime environment is supported by Java Virtual Machine(JVM).

#### 1. JAVA Development Kit (JDK)

The JDK kit is a collection of tools which are used for developing designing, debugging , executing and running JAVA programs. JDK kit includes

- **appletviewer (for viewing JAVA applets) –**

It enables to run JAVA applets (without actually using a JAVA-compatible browser)

- **javac (JAVA compiler)–**

JAVA compiler is used to compile JAVA files. JAVA compiler components of JDK is accessed using “javac” command.

E.G. – C:\>JAVAc filename.JAVA

- **java (java interpreter)**

JAVA interpreter is used to interpret the JAVA files that are compiled by JAVA compiler. JAVA interpreter components of JDK is accessed using “JAVA” command.

E.G. – C:\>JAVA filename

- **javap (JAVA disassemble) –**

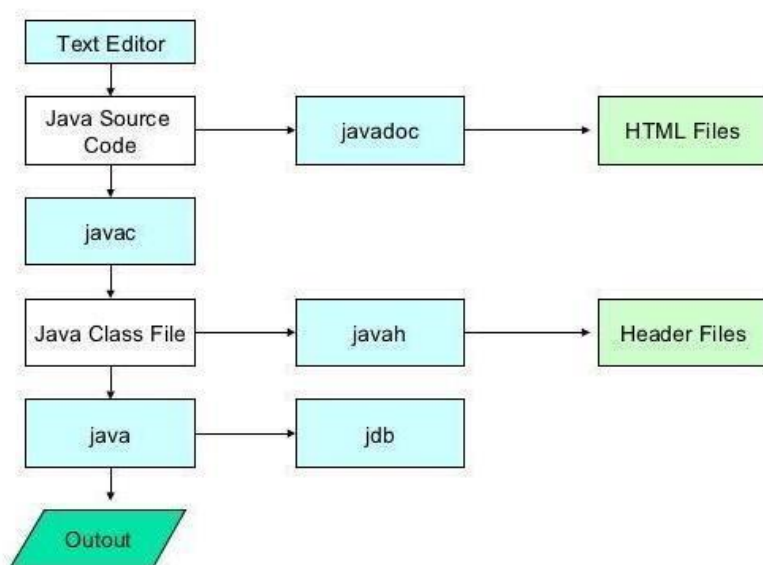
JAVA disassembler, which enables to convert bytecode files into a program description.

- **javah (for C header files) –** produces header files for use with native methods

- **Javadoc (for creating HTML documents) –** creates html-format documentation from JAVA source code files

- **jdb (JAVA debugger) –** JAVA debugger, which helps us to find errors in our program

The following Fig. describes the tools used in JAVA environment.



*Fig. Execution process of java application program*

## 2. JAVA Virtual Machine(JVM) & Byte Code

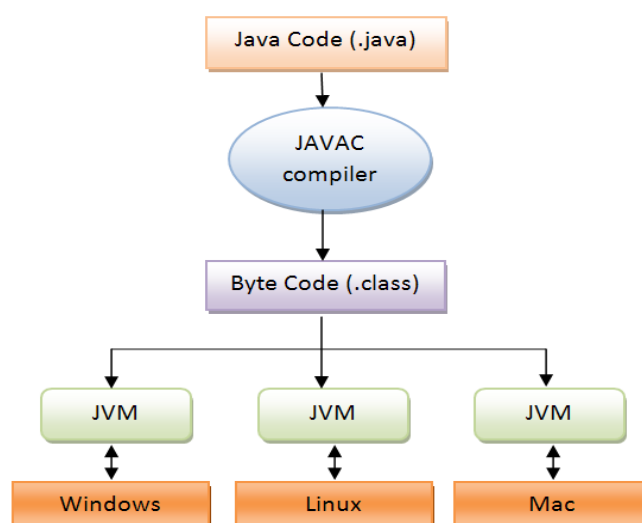
- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

### ByteCode

Java bytecode is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.





---

When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code. When we wish to run this .class file on any other platform, we can do so. After the first compilation, the bytecode generated is now run by the Java Virtual Machine and not the processor in consideration. This essentially means that we only need to have basic java installation on any platforms that we want to run our code on. Resources required to run the bytecode are made available by the Java Virtual Machine, which calls the processor to allocate the required resources. JVM's are stack-based so they stack implementation to read the codes.

### 1.1.3. Simple Java Programming

How to Write and compile the Simple JAVA Program?

For executing any JAVA program, one need to

- Install the JDK.
- Set path of the jdk/bin directory
- Compile and run the JAVA program

After installing JDK, set the path using following command,

Right click on My computer -> properties -> environmental variable -> user variable

New- variable name- 'Path', variable value – "C:\Program Files\JAVA\jdk1.6.0\bin"

To create a simple JAVA program, one need to create a class that contains main method. Open notepad Write this program.

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

Save this file as Simple.java

To compile – javac Filename.java // create Filename.class file

To execute – java Classname // Display output

Open Command Prompt to Compile & Execute JAVA program

```
C:\Program Files\JAVA\jdk1.6.0\bin> javac Simple.JAVA // Simple.class Created
```

```
C:\Program Files\JAVA\jdk1.6.0\bin> java Simple
```



---

Hello JAVA

## Understanding first JAVA program

Meaning of class, public, static, void, main, String[ ], System.out.println().

- **class** keyword is used to declare a class in JAVA
- **public** keyword is an access modifier which represents visibility, it means it is visible to all
- **static** is a keyword, if one declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory
- **void** is the return type of the method, it means it doesn't return any value
- **main** represents startup of the program
- **String[] args** is used for command line argument. This is explained later
- **System.out.println()** is used print statement. The internal working of System.out.println statement is explained later

### Different codes to write the main method

- `public static void main(String[ ] args)`
- `public static void main(String [ ]args)`
- `public static void main(String args[ ])`
- `public static void main(String... args)`

## 1.2. Defining a class, creating object and accessing class members

### 1.2.1. Class

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

A class in JAVA can contain

- data member
- method
- constructor
- block
- class and interface

### Syntax to declare a class

```
class <class_name>
{
```





```
    data member;
    method;
}
```

### A simple class example

```
class student
{
    String name;
    int rollno;
    int age;
}
```

When a reference is made to a particular student with its property then it becomes an object, physical existence of Student class.

```
student std=new student();
```

After the above statement std is instance/object of Student class. Here the new keyword creates an actual physical copy of the object and assign it to the std variable.

### 1.2.2. Object

An entity that has state and behavior is known as an object E.G. chair, bike, marker, pen, table, car etc. It can be physical or logical.

### Simple Example of Object and Class

In this example, a Student class is creator that have two data members viz. id and name. The object of the Student class is created by new keyword and printing the objects value.

```
class Student
{
    int id; //data member (also instance variable)
    String name; //data member(also instance variable)
    public static void main(String args[])
    {
        Student s1=new Student1(); //creating an object of Student
        System.out.println(s1.id);
        System.out.println(s1.name);
    }
}
```



---

This object will produce the following result

0 null

### 1.2.3. new keyword

The new keyword is used to allocate memory at runtime.

### Object and class that maintains the records of students

#### Example-1

In this example, the two objects of Student class are created and initializing the value to these objects by invoking the insertRecord method on it. Here, the state (data) of the objects are displayed by invoking the display Information method.

```
class Student
{
    int rollno;
    String name;
    void insertRecord(int r, String n) //method
    {
        rollno=r;
        name=n;
    }
    void displayInformation( ) //method
    {
        System.out.println(rollno+" "+name);
    }
    public static void main(String args[])
    {
        Student s1=new Student2();
        Student s2=new Student2();
        s1.insertRecord(111,"Riya");
        s2.insertRecord(222,"Amol");
        s1.displayInformation();
        s2.displayInformation();
    }
}
```



---

This will produce the following result

```
111 Riya
222 Amol
```

### Example-2 of Object and Class

There is given another example that maintains the records of Rectangle class. Its explanation is same as in the above Student class example.

class Rectangle

```
{
    int length;
    int width;
    void insert(int l, int w)
    {
        length=l;
        width=w;
    }
    void calculateArea()
    {
        System.out.println(length*width);
    }
    public static void main(String args[])
    {
        Rectangle r1=new Rectangle();
        Rectangle r2=new Rectangle();
        r1.insert(11,5);
        r2.insert(3,15);
        r1.calculateArea();
        r2.calculateArea();
    }
}
```

This will produce the following result



---

55

45

#### 1.2.4. Method in JAVA

Method describe behavior of an object. A method is a collection of statements that are group together to perform an operation.

##### Syntax of method is

```
return-type methodName(parameter-list)
{
    //body of method
}
```

##### Example of a Method

```
public String getName(String st)
{
    String name="CoreJava";
    name=name+st;
    return name;
}
```

#### 1.3. JAVA Tokens

Tokens are the various JAVA program elements which are identified by the compiler. A token is the smallest element of a program that is meaningful to the compiler. Tokens supported in JAVA include keywords, variables, constants, special characters, operations etc.

Tokens are the smallest unit of Program. There is Five Types of Tokens

- Reserve Word or Keywords
- Identifier
- Literals
- Operators
- Separators

##### 1.3.1. JAVA Key Words

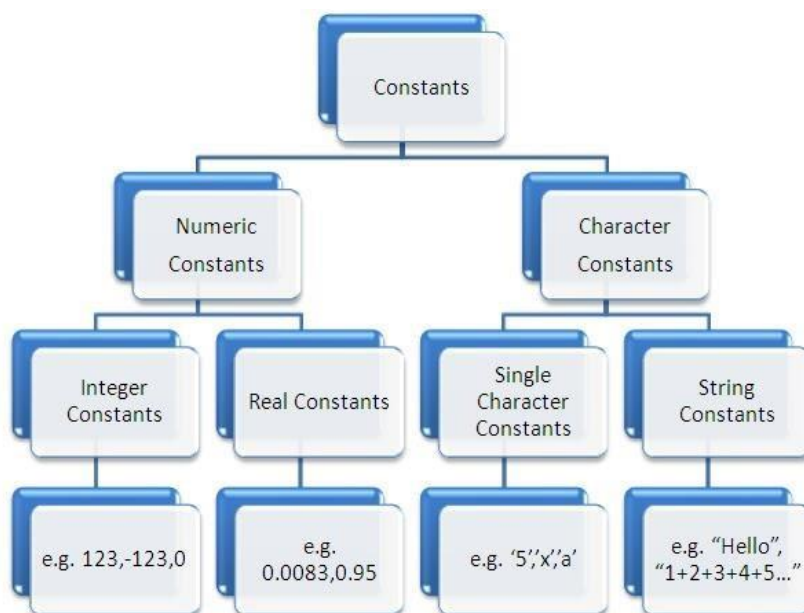
In the JAVA programming language, a keyword is one of 50 reserved words. that have a predefined meaning in the language; because of this, programmers cannot use keywords as names for variables, methods, classes, or as any other identifier.

abstract	assert	boolean	break	byte	case
catch	char	class	const*	continue	default
double	do	else	enum	extends	false
final	finally	float	for	goto*	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient
true	try	void	volatile	while	

Even though “goto” and “const” are no longer used in the JAVA programming language, they still cannot be used.

### 1.3.2. Constants

Constants in JAVA are fixed values those are not changed during the Execution of program JAVA supports several types of Constants those are



#### Integer Constants

Integer Constants refers to a Sequence of digits which Includes only negative or positive Values and many other things those are as follows

- An Integer Constant must have at Least one Digit
- It must not have a Decimal value
- It could be either positive or Negative
- If no sign is Specified then it should be treated as Positive
- No Spaces and Commas are allowed in Name



## Real Constants

- A Real Constant must have at Least one Digit
- It must have a Decimal value
- It could be either positive or Negative
- If no sign is Specified then it should be treated as Positive
- No Spaces and Commas are allowed in Name
- Like 251, 234.890 etc are Real Constants

In The Exponential Form of Representation the Real Constant is Represented in the two Parts The part before appearing e is called mantissa whereas the part following e is called Exponent.

- In Real Constant The Mantissa and Exponent Part should be Separated by letter “e”
- The Mantissa Part have may have either positive or Negative Sign
- Default Sign is Positive

## Single Character Constants

A Character is Single Alphabet a single digit or a Single Symbol that is enclosed within Single inverted commas.

Like ‘S’ ,’1’ etc are Single Character Constants

## String Constants

String is a Sequence of Characters Enclosed between double Quotes These Characters may be digits , Alphabets Like “Hello” , “1234” etc.

## Backslash Character Constants

JAVA Also Supports Backslash Constants. These are used in output methods. E.G. – \n is used for new line Character These are also Called as escape Sequence or backslash character Constants.

E.G.

\t For Tab ( Five Spaces in one Time )

\b Back Spac

\f form feed

\n line feed

\r carriage reurn

\” double quote

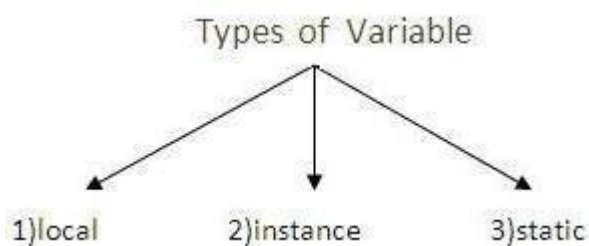
\’ single quote

\\ blackslash

### 1.3.3. Variable

Variable is name of reserved area allocated in memory.

E.G. – int data = 50; // Here data is variable



There are three types of variables in JAVA

- **Local variable** – a variable that is declared inside the method is called local variable
- **Instance variable** – a variable that is declared inside the class but outside the method is called instance variable . It is not declared as static
- **Static variable** – a variable that is declared as static is called static variable. It cannot be local

#### Variable declarations example

```
int maxAge;  
int x, y, selectedIndex;  
char a, b;  
boolean flag;  
double maxVal, massInKilos;
```

#### Variable initialization in declaration example

```
int timeInSeconds = 245;  
char a = 'K', b = '$';  
boolean flag = true;  
double maxVal = 35.875;
```

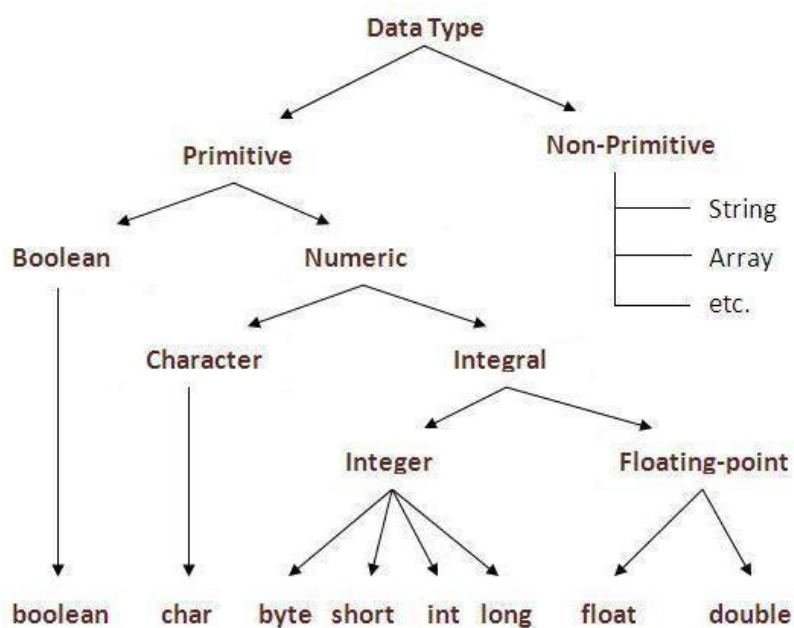
#### Types of variables example

```
class A  
{  
    int data=50;           //instance variable  
    static int m=100;     //static variable  
    void method()  
    {  
        int n=90;        //local variable  
    }  
}                          //end of class
```

### 1.3.4. Data Types in JAVA

In JAVA, there are two types of data types

- Primitive data types
- Non-primitive data types



Data Type	Default Value	Default size
boolean	FALSE	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

## String Objects

A string in JAVA is not a primitive data type. It is an object from the system defined class String.

## String constants example

"watermelon", "fig", "\$%&\*^%!!", "354", " " (space), "" (null string)

## The string declaration example

```
String item = "apple";
```

This declaration is short for the object declaration.

```
String item = new String("apple");
```





## Arrays

An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.

### 1.4. Operator and Expression

Operators are special symbols used for mathematical functions, some types of assignment statements, and logical comparisons.

An expression is a statement that can convey a value. Some of the most common expressions are mathematical E.G.

```
int x = 3;
int y = x;
int z = x * y
```

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

1. Arithmetic Operators
2. Increment & Decrement Operators
3. Relational Operators
4. Bitwise Operators
5. Logical Operators
6. Assignment Operators

#### 1.4.1. Arithmetic Operators

Arithmetic operators are used in mathematical expressions.

Considered two variables A & B

Operator	Meaning	Syntax
+	Addition	A + B
-	Subtraction -	A - B
*	Multiplication -	A * B
/	Division -	A / B
%	Modulus	A% B
++	Increment - Increases the value of operand by 1	A++ ,
--	Decrement - Decreases the value of operand by 1	A --



---

**The following Program Demonstrate the use of Basic arithmetic operators**

```
public class ArithmDemo
{
    public static void main(String[] args)
    {
        int add = 2 + 4;
        System.out.println( "2 + 4 = " + add);
        int subtract = 10 - 3;
        System.out.println("10 - 3 = " + subtract);
        int divide = 6/2;
        System.out.println("6 / 2 = " + divide);
        int multiply = 5 * 5;
        System.out.println("5 * 5 = " + multiply);
        int modulus = 7 % 2;
        System.out.println("7 % 2 = " + modulus);
    }
}
```

The output from above program will be

```
2 + 4 = 6
10 - 3 = 7
6 / 2 = 3
5 * 5 = 25
7 % 2 = 1
```

### 1.4.2. Increment and Decrement Operators

The increment operator increases its operand by one. The decrement operator decreases its operand by one.

E.G.

```
x = x + 1;
```

The above statement can be rewritten by using the increment operator

```
x++;
```

Similarly, the statement

```
x = x - 1
```

is equivalent to

```
x--;
```

**The following program demonstrates the increment operator.**

```
class IncDec
{
```



```
public static void main(String args[])
{
    int a = 1;
    int b = 2;
    int c;
    int d;
    c = ++b;
    d = a++;
    c++;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("d = " + d);
}
}
```

### 1.4.3. Relational Operators

The relational operators determine the relationship that one operand has to the other.

Operator	Meaning	Syntax
==	Equal to	(A == B)
!=	Not equal to	(A != B)
>	Greater than	(A > B)
<	Less than	(A < B)
>=	Greater than or equal to	(A >= B)
<=	Less than or equal to	(A <= B)

Following program demonstrate the use of Relational Operator ==, !=, >, <, >= and <=

```
class RelOpTrDemo
```

```
{
    public static void main(String[] args)
    {
        int a = 10, b = 15, c = 15;
        System.out.println("Relational Operators and returned values");
        System.out.println(" a > b = " + (a > b));
    }
}
```



```
System.out.println(" a < b = " + (a < b));
System.out.println(" b >= a = " + (b >= a));
System.out.println(" b <= a = " + (b <= a));
System.out.println(" b == c = " + (b == c));
System.out.println(" b != c = " + (b != c));
}
}
```

Output from above program will be

```
a > b = false
a < b = true
b >= a = true
b <= a = false
b == c = true
b != c = false
```

#### 1.4.4. Bitwise Operators

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format, value of a & b will be

```
a = 0011 1100
b = 0000 1101
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011
```

Operator	Description	Syntax
&	Bitwise AND	(A & B)
	Bitwise OR	(A   B)
^	Bitwise exclusive OR	(A ^ B)
~	Bitwise unary NOT	(~A).
<<	Shift left	A << 2
>>	Shift right	A >> 2
>>>	Shift right zero fill	A >>>2 will give 15 which is 0000 1111



---

## The following is the Demonstrating example of Bitwise Operators

```
public class Test
{
    public static void main(String args[])
    {
        int a = 60; /* 60 = 0011 1100 */
        int b = 13; /* 13 = 0000 1101 */
        int c = 0;
        c = a & b; /* 12 = 0000 1100 */
        System.out.println("a & b = " + c);
        c = a | b; /* 61 = 0011 1101 */
        System.out.println("a | b = " + c);
        c = a ^ b; /* 49 = 0011 0001 */
        System.out.println("a ^ b = " + c);
        c = ~a; /* -61 = 1100 0011 */
        System.out.println("~a = " + c);
        c = a << 2; /* 240 = 1111 0000 */
        System.out.println("a << 2 = " + c);
        c = a >> 2; /* 15 = 0000 1111 */
        System.out.println("a >> 2 = " + c);
        c = a >>> 2; /* 15 = 0000 1111 */
        System.out.println("a >>> 2 = " + c);
    }
}
```

Output from above program will be

```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 2 = 15
a >>> 2 = 15
```



### 1.4.5. Logical Operators

Assume Boolean variables A holds true and variable B holds false then the following table lists the logical operators

Operator	Description	Syntax
&&	Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A    B) is true.
!	Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

**The following program Demonstrates the use of logical operators.**

```
public class Test
{
    public static void main(String args[])
    {
        boolean a = true;
        boolean b = false;
        System.out.println("a && b = " + (a&&b));
        System.out.println("a || b = " + (a||b) );
        System.out.println("!(a && b) = " + !(a && b));
    }
}
```

Output from above program will be

```
a && b = false
a || b = true
!(a && b) = true
```

### 1.4.5. Assignment Operators

Following assignment operators are supported by JAVA



Operator	Description	Syntax/Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ will assign value of $A + B$ into $C$
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$C %= A$ is equivalent to $C = C \% A$
<<=	Left shift AND assignment operator	$C <<= 2$ is same as $C = C << 2$
>>=	Right shift AND assignment operator	$C >>= 2$ is same as $C = C >> 2$
&=	Bitwise AND assignment operator	$C \&= 2$ is same as $C = C \& 2$
^=	bitwise exclusive OR and assignment operator	$C \wedge= 2$ is same as $C = C \wedge 2$
=	bitwise inclusive OR and assignment operator	$C  = 2$ is same as $C = C   2$

**The following program demonstrates the assignment operators**

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



---

```
int a = 10;
int b = 20;
int c = 0;
c = a + b;
System.out.println("c = a + b = " + c );
c += a ;
System.out.println("c += a = " + c );
c -= a ;
System.out.println("c -= a = " + c );
c *= a ;
System.out.println("c *= a = " + c );
a = 10;
c = 15;
c /= a ;
System.out.println("c /= a = " + c );
}
}
```

Output from above program will be

```
c = a + b = 30
c += a = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a = 0
c ^= a = 10
c |= a = 10
```





## Special Operators

There are few other operators supported by JAVA Language.

### 1.4.6 Conditional Operator ( ? : )

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.

The operator is written as follows

variable x = (expression) ? value if true : value if false

#### The following program demonstrates the conditional operator

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

Output from above program will be

Value of b is : 30

Value of b is : 20

### 1.4.7. instanceof Operator

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type).

instanceof operator is written as

( Object reference variable ) instanceof ( class/interface type)

#### The following program demonstrates the instanceof operator

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



```
{  
    String name = "James"; // following will return true since name is type of String  
    boolean result = name instanceof String;  
    System.out.println( result );  
}  
}
```

Output from above program will be

true

### 1.4.8. Precedence of JAVA Operators

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

E.G. – the multiplication operator has higher precedence than the addition operator

E.G. –  $x = 7 + 3 * 2$ ; here x is assigned 13, not 20 because operator \* has higher precedence than +, so it first gets multiplied with  $3*2$  and then adds into 7.

Category	Operator	Associativity
Postfix	( ) [ ] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	? :	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^=  =	Right to left
Comma	,	Left to right



### 1.4.9. Mathematical functions

- Java provides a support for performing mathematical operations by means of mathematical functions.
- The basic mathematical functions are defined in the **math** class defined in **java.lang** package.
- Various mathematical functions are

sin(double angle)	Returns the sine of angle in radians
cos(double angle)	Returns the cosine of angle in radians
tan(double angle)	Returns the tan of angle in radians
asin(double val)	Returns the angle whose sine is val
acos(double val)	Returns the angle whose cosine is val
atan(double val)	Returns the angle whose tan is val
pow(double a, double b)	It computes $a^b$
exp(double a)	It computes $e^a$
sqrt(double a)	It computes square root of a
max(a,b)	It computes the maximum of a & b
min(a,b)	It computes the minimum of a & b
log(val)	It computes the logarithm values of val
abs(val)	It computes the absolute value of val
ceil(val)	It returns the smallest whole no. which is greater than or equal to val
floor(val)	It returns the largest whole no. which is lesser than or equal to val

#### Program for demonstrating mathematical functions

```
class MathDemo
{
    public static void main(String args[])
    {
        double val=25;
        System.out.println("The square root of " +val+ " is" +Math.sqrt(val));
        int a=10, b=20;
        System.out.println("The maximum of " +a+ " and" +b+ " is"+Math.max(a,b));
        System.out.println("The minimum of " +a+ " and" +b+ " is"+Math.min(a,b));
    }
}
```



---

```
        System.out.println("The sine of " +val+ "is" +Math.sin(val));
        System.out.println("The cosine of " +val+ "is" +Math.cos(val));
    }
}
```

Output from above program will be

```
The square root of 25.0 is 5.0
The maximum of 100 and 20 is 20
The minimum of 10 and 20 is 10
The sin of 25.0 is -0.13235175009777303
The cos of 25.0 is 0.9912028118634736
```

## 1.5. Decision making and looping

JAVA language supports two types of selections statements

- if statements
- switch statements

### 1.5.1. Decision Making with If Statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

- Simple if statement
- if... else statement
- Nested if...else statement
- else if ladder

#### 1.5.1.1. Simple If Statement

The syntax of the “if statement” is shown below

```
if(expression)
    statements;
```

Where a statement may consist of a single statement, a compound statement, or nothing. The expression must be enclosed in parentheses. If the expression evaluates to true, the statement is executed, otherwise ignored.

**The following program demonstrates the if statement**

class Test

```
{
```



---

```
public static void main(String args[])
{
    int x = 10;
    if( x < 20 )
    {
        System.out.print("This is if statement");
    }
}
```

This would produce the following result

This is if statement

### 1.5.1.2. The If... Else Statement

This form of if allows either-or condition by providing an else clause.

The syntax of the if-else statement is the following

```
if(expression)
    statemen-1;
else
    statement-2;
```

If the expression evaluates to true i.e., a non-zero value, the statement-1 is executed, otherwise statement-2 is executed. The statement-1 and statement-2 can be a single statement, or a compound statement, or a null statement.

### The following program demonstrates the if...else statement

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



```
        {  
            System.out.print("This is else statement");  
        }  
    }  
}
```

This would produce the following result

This is else statement

### 1.5.1.3. Nested if... else Statement

A Nested if is an if that has another if in its 'if's' body or in its else's body. The nested if can have one of the following three forms

#### Example 1

```
If (expression1)  
    {  
        If (expression2)  
            statement-1;  
        else  
            statement-2;  
    }  
else  
    body of else;
```

#### Example 2

```
If (expression1)  
    body-of-if;  
else  
    {  
        If (expression2)  
            statement-1;  
        else  
            statement-2;  
    }
```

#### Example 3

```
If (expression1)
```



```
{
    If (expression2)
        statement-1;
    else
        statement-2;
}
else
{
    If (expression3)
        statement-3;
    else
        statement-4;
}
```

**The following program demonstrates the nested if...else statement**

public class Test

```
{
    public static void main(String args[])
    {
        int x = 30;
        int y = 10;
        if ( x == 30 )
        {
            If ( y == 10 )
            {
                System.out.print("X = 30 and Y = 10");
            }
        }
    }
}
```

This would produce the following result

X = 30 and Y = 10



---

#### 1.5.1.4. The else-if Ladder

A common programming construct in the JAVA is the if-else-if ladder , which is often also called the if-else-if staircase because of it's appearance.

It takes the following general form

```
if (expression1)
    statement1;
else
    if (expression2)
        statement2;
    else
        if (expression3)
            statement3;
        else
            statement n;
```

**The following program demonstrates the else-if ladder**

```
public class Test
{
    public static void main(String args[])
    {
        int x = 30;
        if ( x == 10 )
        {
            System.out.print("Value of X is 10");
        }
        else
        If ( x == 20 )
        {
            System.out.print("Value of X is 20");
        }
        else
        if ( x == 30 )
        {
```





---

```
        System.out.print("Value of X is 30");
    }
    else
    {
        System.out.print("This is else statement");
    }
}
}
```

This would produce the following result

Value of X is 30

### 1.5.2. Switch Statement

JAVA provides a multiple branch selection statement known as switch.

The syntax of enhanced for loop is

Switch (expression)

```
{
    case value :
        //Statements
    break;    //optional
    case value :
        //Statements
    break;    //optional
    //One can have any number of case statements.
    default : //Optional
        //Statements
}
```

**The following program demonstrates the switch statement**

class SwitchDemo

```
{
    public static void main(String[] args)
    {
        int month = 8;
        switch (month)
```



---

```
{
    case 1 : System.out.println("January");
    break;
    case 2 : System.out.println("February");
    break;
    case 3 : System.out.println("March");
    break;
    case 4 : System.out.println("April");
    break;
    case 5 : System.out.println("May");
    break;
    case 6 : System.out.println("June");
    break;
    case 7 : System.out.println("July");
    break;
    case 8 : System.out.println("August");
    break;
    case 9 : System.out.println("September");
    break;
    case 10 : System.out.println("October");
    break;
    case 11 : System.out.println("November");
    break;
    case 12 : System.out.println("December");
    break;
    default : System.out.println("Invalid month.");
    break;
}
}
```

In this case, "August" is printed to standard output.



---

### 1.5.2.1.Nested Switch

One can use switch as part of the statement sequence of an outer switch. This is called a nested switch. Since a switch statement defines in its own block, no conflicts arise between the case statements in the inner switch and those in the outer switch.

```
switch(count)
{
    case 1 :
        switch(target)
        {
            case 0 :
                System.out.println("target is zero");
                break;
            case 1 :
                System.out.println("target is one");
                break;
        }
        break;
    case2 : //...
}
```

Here the case 1 – statement in inner switch does not conflict with the case 1 : statement in the outer switch. The count variable is only compared with the list of cases at the outer level. if the count is 1, then target is compared with the inner list cases.

*No two case constants in the same switch can have individual values. A switch statement enclosed by an outer switch can have case constants in common.*

### 1.5.3. The while Statement

The most simple and general looping structure available in JAVA is the while statement. The syntax of while loop is

```
while(condition)
{
    // loop-body
}
```



---

**The following program demonstrates the while statement**

```
public class Test
{
    public static void main(String args[])
    {
        int x = 10;
        while ( x < 20 )
        {
            System.out.print("value of x : " + x );
            x++;
            System.out.print("\n");
        }
    }
}
```

This would produce the following result

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

#### **1.5.4. The do while Statement**

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

The syntax of the do-while loop is

```
do
{
```



---

```
    loop-body;  
}
```

```
While (condition);
```

**The following program demonstrates the do-while statement**

```
public class Test  
{  
    public static void main(String args[])  
    {  
        int x = 10;  
        do  
        {  
            System.out.print("value of x : " + x );  
            x++;  
            System.out.print("\n");  
        }  
        While ( x < 20 );  
    }  
}
```

This would produce the following result

```
value of x : 10  
value of x : 11  
value of x : 12  
value of x : 13  
value of x : 14  
value of x : 15  
value of x : 16  
value of x : 17  
value of x : 18  
value of x : 19
```



---

### 1.5.5. The for Statement

The **for** loop is the easiest to understand of the JAVA loops. All its loop-control elements are gathered in one place (on the top of the loop), while in the other loop construction of C++ , they( top-control elements) are scattered about the program.

The Syntax of the for loop statement is

```
for( initialization expression(s); test condition; update expression)
{
    loop-body;
}
```

#### //program of for loop

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

This would produce the following result

```
1
2
..
.
.
10
```

### 1.5.6. The break Keyword

The **break** keyword is used to stop the entire loop. The **break** keyword must be used inside any loop or a switch statement. The **break** keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

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



---

```
break;
```

**The following program demonstrates the break keyword**

```
public class Test
{
    public static void main(String args[])
    {
        int [ ] numbers = {10, 20, 30, 40, 50};
        for (int x : numbers )
        {
            If ( x == 30 )
            {
                break;
            }
            System.out.print( x );
            System.out.print("\n");
        }
    }
}
```

This would produce the following result

```
10
20
```

### 1.5.7. The continue Keyword

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

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

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

```
continue;
```



---

**The following program demonstrates the continue keyword**

```
public class Test
{
    public static void main(String args[])
    {
        int [ ] numbers = {10, 20, 30, 40, 50};
        for (int x : numbers )
        {
            If ( x == 30 )
            {
                continue;
            }
            System.out.print( x );
            System.out.print("\n");
        }
    }
}
```

This would produce the following result

```
10
20
40
50
```

## 1.6. Programs

- 1) Write a program in java to print your name?
- 2) Program for creating class students and printing the data for two objects?
- 3) Write a program in java to demonstrate the arithmetic operators?
- 4) Write a program in java, to calculate the area and circumference of circle?
- 5) Write a program in java to calculate the area and perimeter of square?
- 6) Write a program in java calculate the area and perimeter of rectangle?
- 7) Write a program in java to demonstrate the math functions?
- 8) Write a program in java to find the large/small no among the two nos?





- 
- 9) Write a program in java to check the no odd or even?
  - 10) Write a program in java to find the large/small no among the three nos?
  - 11) Write a program in java to check ovel or consonants?
  - 12) Write a program in java to reverse the given nos?
  - 13) Write a program in java to print the multiplication table of 10?
  - 14) Write a program in java to check the no is divisible by 7?
  - 15) Write a program in java to print your name five times?
  - 16) Write a program in java to find the sum of digit of given no?
  - 17) Write a program in java to find the no of and sum of all integers greater than 100 and less than 200 that are divisible by7.
  - 18) Write a program in java to find all the odd nos between 120 to 210?
  - 19) Write a program in java to generates the fibbonacci series of given no?
  - 20) Write a program in java to find the factorial no of given no?
  - 21) Write a program in java to check the prime no?
  - 22) Write a program in java to print the following outputs?

a)	b)	c)	d)	e)
1	1	*	1 1 1 1 1	*
1 2	2 2	**	2 2 2 2	***
1 2 3	3 3 3	***	3 3 3	*****
1 2 3 4	4 4 4 4	****	4 4	*****
1 2 3 4 5	5 5 5 5 5	*****	5	*****

**End of Unit-I**

---



## 2.1. Constructor and methods

### 2.1.1. Methods

- Method describe behavior of an object.
- A method is a collection of statements that are group together to perform an operation.

Syntax of method is

```
return-type methodName(parameter-list)
{
    //body of method
}
```

Example of a Method

```
public String getName(String st)
{
    String name="Java Programming";
    name=name+st;
    return name;
}
```

```
public String getName(String st)
    ↑      ↑      ↑      ↑
modifier return-type method-name parameter
```

**Modifier** – Modifier are access type of method.

**Return Type** – A method may return value. Data type of value return by a method is declare in method heading

**Method name** – Actual name of the method

**Parameter** – Value passed to a method

**Method body** – collection of statement that defines what method does

### 2.1.2. Constructor

- Constructor in JAVA is a special type of method that is used to initialize the object.
- JAVA constructor is invoked at the time of object creation.
- It constructs the values i.e. provides data for the object that is why it is known as constructor.
- A constructor has same name as the class in which it resides.
- Constructor in JAVA cannot be abstract, static, final or synchronized.
- These modifiers are not allowed for constructor.



```
class Car
{
    String name ;
    String model;
    Car() //Constructor
    {
        name ="";
        model="";
    }
}
```

### Rules for creating JAVA constructor

There are basically two rules defined for the constructor

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

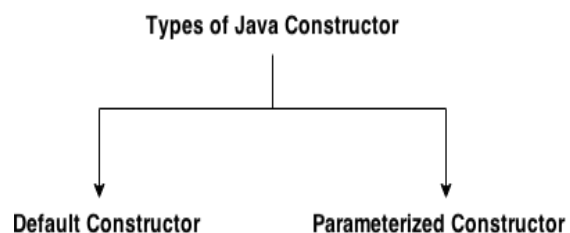
### Difference between Constructor and Method

Sr.No	Constructor	Method
1	The name of the constructor must be same as the class name.	The name of the method should not be the class name.
2	It does not return anything hence no return type.	It can return and hence it has a return type. If method does not return anything then the return type is void.
3	The purpose of constructor is to initialize the data members of the class.	The method is defined to execute the core logic of the class.
4	The constructor is invoked implicitly at the time of object creation.	The method must be called explicitly using the object name and dot operator.

### 2.1.3. Types of JAVA constructors

There are two types of constructors

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





## 1. Default Constructor

- A constructor that have no parameter is known as default constructor.

Syntax of default constructor

```
<class_name> ( )  
{  
}
```

### Example of default constructor.

```
class Bike1  
{  
    Bike1()  
    {  
        System.out.println("Bike is created");  
    }  
    public static void main(String args[ ])  
    {  
        Bike1 b=new Bike1();  
    }  
}
```

This will produce the following result

Bike is created

**Note – If there is no constructor in a class, compiler automatically creates a default constructor.**

### Example of default constructor that displays the default values

```
class Student  
{  
    int id;  
    String name;  
    void display( )  
    {  
        System.out.println(id+" "+name);  
    }  
    public static void main(String args[])
```



```
{
    Student s1=new Student();
    Student s2=new Student();
    s1.display();
    s2.display();
}
}
```

This will produce the following result

0 null

0 null

In the above class, any constructor is not created any constructor. Compiler provides the default constructor. Here 0 and null values are provided by default constructor.

## 2. Parameterized Constructor

- A constructor that have parameters is known as parameterized constructor.
- Parameterized constructor is used to provide different values to the distinct objects.

### Example of parameterized constructor.

```
class Student
{
    int id;
    String name;
    Student(int i, String n)
    {
        id = i;
        name = n;
    }
    void display( )
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[ ])
    {
```



```
Student s1 = new Student4(111,"Ram");
Student s2 = new Student4(222,"Shyam");
s1.display();
s2.display();
}
}
```

This will produce the following result

```
111 Ram
222 Shyam
```

#### 2.1.4. Arguments Passing

- There are two ways to pass an argument to a method.
  1. call-by-value
  2. call-by-reference

**NOTE** – In JAVA, when a primitive type is passed to a method, it is passed by value whereas when an object of any type is passed to a method, it is passed as reference.

##### 1. call-by-value

- In this approach copy of an argument value is pass to a method. Changes made to the argument value inside the method will have no effect on the arguments.

##### Example of call-by-value

```
public class Test
{
    public void callByValue(int x)
    {
        x=100;
    }
    public static void main(String[] args)
    {
        int x=50;
        Test t = new Test();
        t.callByValue(x); //function call
        System.out.println(x);
    }
}
```



```
}
```

This will produce the following result

```
50
```

## 2. call-by-reference

- In this reference of an argument is pass to a method. Any changes made inside the method will affect the argument value

### Example of call-by-reference

```
public class Test
{
    int x=10;
    int y=20;
    public void callByReference(Test t)
    {
        t.x=100;
        t.y=50;
    }
    public static void main(String[] args)
    {
        Test ts = new Test();
        System.out.println("Before "+ts.x+" "+ts.y);
        ts.callByReference(ts);
        System.out.println("After "+ts.x+" "+ts.y);
    }
}
```

This will produce the following result

```
Before 10 20
```

```
After 100 50
```

### 2.1.5. this keyword

- this keyword is used to refer to current object.
- this is always a reference to the object on which method was invoked.
- this can be used to invoke current class constructor.
- this can be passed as an argument to another method.



---

### Example

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

Here this keyword is used to initialize member of current object.

### Program for this Keyword

```
class thisDemo
{
    int a, b;
    public void sum(int a, int b)
    {
        this.a=a;
        this.b=b;
    }
    public void show()
    {
        int c=a+b;
        System.out.println("a="+a);
        System.out.println("b="+b);
        System.out.println("sum="+c);
        System.out.println("-----");
    }
    public static void main(String args[])
    {
        thisDemo d1= new thisDemo();
    }
}
```





```
        d1.sum(10,20);
        d1.show();
        thisDemo d2= new thisDemo();
        d2.sum(20,30);
        d2.show();
    }
}
```

Output:-

a=10

b=20

sum=30

-----

a=20

b=30

sum=50

### 2.1.6. Command line argument in JAVA

- The command line argument is the argument passed to a program at the time when it is run.
- To access the command-line argument inside a JAVA program is quite easy, they are stored as string in String array passed to the args parameter of main() method.

Example

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

Execute this program a



```
C:\>javac cmddemo.java
```

```
C:\>java cmddemo 10 Gunwant 30
```

This will produce the following result

```
10
```

```
Gunwant
```

```
30
```

## Programs on Command Line arguments

- 1) Write a program to accept two number as command line arguments and print addition of those number?

```
class demo
{
    public static void main(String args[])
    {
        int n1=Integer.parseInt(args[0]);
        int n2=Integer.parseInt(args[1]);
        int add=n1+n2;
        System.out.println(add);
    }
}
```

```
C:\>javac demo.java
```

```
C:\>java demo 10 20
```

```
30
```

- 2) Write a program to accept number from command line and print the number is odd or even?

```
class clsdemo
{
    public static void main(String[] args)
    {
        int n=Integer.parseInt(args[0]);
        if(n%2==0)
        {
```



---

```
        System.out.println(n+" Is a even no");
    }
    else
        System.out.println(n+" Is Odd no");
    }
}
```

```
C:\>javac clsdemo.java
```

```
C:\>java clsdemo 10
```

```
10 Is even no
```

### 3) Write a program to accept number from command line and print square root of the number?

```
class clsdemo3
{
    public static void main(String[] args)
    {
        int n= Integer.parseInt(args[0]);
        double ans;
        ans=Math.sqrt(n);
        System.out.println("Square root of "+n+"= " +ans);
    }
}
```

```
>javac clsdemo3.java
```

```
>java clsdemo3 25
```

```
Square root of 25= 5.0
```

#### 2.1.7. Garbage collection

- In JAVA destruction of object from memory is done automatically by the JVM.
- When there is no reference to an object, then that object is assumed to be no longer needed and the memory occupied by the object are released.
- This technique is called Garbage Collection.
- This is accomplished by the JVM.
- Unlike C++ there is no explicit need to destroy object.



### 2.1.8. finalize( ) method

- Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held.
- To handle such situation finalize() method is used.
- finalize()method is called by garbage collection thread before collecting object.
- Its the last chance for any object to perform cleanup utility.

Signature of finalize() method

```
protected void finalize()
```

```
{  
    //finalize-code  
}
```

Some Important Points to Remember

1. finalize() method is defined in JAVA.lang.Object class, therefore it is available to all the classes
2. finalize() method is declare as protected inside Object class
3. finalize() method gets called only once by GC(Garbage Collector) threads

### 2.1.9. Object class

- In java there is a special class named **Object**
- **Object** is a super class of all other classes by default.
- The object can be obtained using getObject() method.

For example

```
Object obj=getObject();
```

## 2.2. Visibility Control

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.



3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

### Example

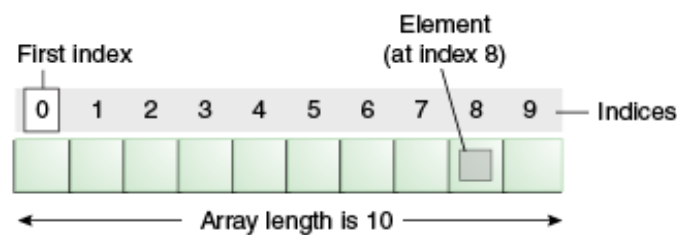
```
class accessModifier
{
    int a; //default
    public int b;
    private int c;
    void fun(int val)
    {
        c=val;
    }
    void show( )
    {
        System.out.println("c= "+c);
    }
}

class Demo
{
    public static void main(String args[])
    {
        accessModifier obj= new accessModifier();
        obj.a=100; //valid
        obj.b=200; //valid
        obj.c=300; //Error: private access
        obj.fun(300); //valid
        obj.show(); //valid
    }
}
```



## 2.3. Arrays

- An array is a collection of similar type of elements which has contiguous memory location.
- Java array is an object which contains elements of a similar data type.
- We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
- Java provides the feature of anonymous arrays which is not available in C/C++.



### Type of Array

There are two types of array.

- One/ Single Dimensional Array
- Two/Multidimensional Array

#### 2.3.1. One/ Single Dimensional Array

The One dimensional array can be represented as

Array a[5]

10	20	70	40	50
0	1	2	3	4

### Example

```
class Testarray
{
    public static void main(String args[])
    {
        int a[ ]=new int[5];    //declaration and instantiation
        a[0]=10;//initialization
        a[1]=20;
        a[2]=30;
        a[3]=40;
        a[4]=50;
        //traversing array
    }
}
```



---

```
        for(int i=0;i<a.length;i++) //length is the property of array
        System.out.println(a[i]);
    }
}
```

Output

```
10
20
70
40
50
```

**We can declare, instantiate and initialize the java array together by:**

```
int a[ ]={33,3,4,5}; //declaration, instantiation and initialization
```

**Let's see the simple example to print this array.**

```
class Testarray1
{
    public static void main(String args[])
    {
        int a[ ]={10,20,30,40,50}; //declaration, instantiation and initialization
        //printing array
        for(int i=0;i<a.length; i++) //length is the property of array
        System.out.println(a[i]);
    }
}
```

Output

```
10
20
30
40
50
```

### 2.3.2. Two dimensional Array

In such case, data is stored in row and column based index (also known as matrix form).

Example to instantiate Multidimensional Array in Java



---

```
int[ ][ ] arr =new int[3][3]; //3 row and 3 column
```

### Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

### Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

//Java Program to illustrate the use of multidimensional array

```
class Testarray2
{
public static void main(String args[])
{
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
System.out.print(arr[i][j] + " ");
}
System.out.println();
}
}
}
```

Output:





---

1 2 3

2 4 5

4 4 5

### 2.3.3. Array of objects

#### Syntax

```
objname[ ]= new classname( );
```

#### Program:-

- **Define a class 'student' with data members stdrn, name and marks. Accept data for five objects using array of objects and print it.**

```
// Reading the data from keyboard
```

```
import java.util.*;
```

```
class student
```

```
{
```

```
    int stdrn;
```

```
    String name;
```

```
    Double marks;
```

```
    char gender;
```

```
    void getdata()
```

```
    {
```

```
        Scanner in=new Scanner(System.in);
```

```
        System.out.println("Enter Student Roll no:");
```

```
        stdrn=in.nextInt();
```

```
        System.out.println("Enter Student Name:");
```

```
        name=in.next();
```

```
        System.out.println("Enter Student Marks:");
```

```
        marks=in.nextDouble();
```

```
        System.out.println("Enter the Gender:");
```

```
        gender=in.next().charAt(0);
```

```
    }
```

```
void display()
```

```
{
```

```
    System.out.println("-----");
```



```
        System.out.println("Student Roll no: "+stdrn);
        System.out.println("Student Name: "+name);
        System.out.println("Student Marks: "+marks);
        System.out.println("Student Gender: "+gender);
        System.out.println("-----");
    }
}
class arrayofobject
{
    public static void main(String[] args)
    {
        student s[ ]=new student[3];
        System.out.println("Enter the Student data");
        for(int i=0; i<3 ;i++)
            {
                s[i]= new student();
                s[i].getdata();
            }
        System.out.println("Student Details is as follows");
        for(int i=0; i<3 ;i++)
            {
                s[i].display();
            }
    }
}
```

**Output:**

Enter the Student data

Enter Student Roll no:

1

Enter Student Name:

rohini

Enter Student Marks:



---

45.67

Enter the Gender:

F

Enter Student Roll no:

2

Enter Student Name:

vaidehi

Enter Student Marks:

78.65

Enter the Gender:

F

Enter Student Roll no:

3

Enter Student Name:

Ashish

Enter Student Marks:

46.65

Enter the Gender:

M

Student is as follows

-----  
Student Roll no: 1

Student Name: rohini

Student Marks: 45.67

Student Gender: F

-----  
Student Roll no: 2

Student Name: vaidehi

Student Marks: 78.65

Student Gender: F  
-----



-----  
Student Roll no: 3

Student Name: Ashish

Student Marks: 46.65

Student Gender: M  
-----

## 2.4. Strings

Definition- String is a collection of characters.

In java, the string can be defined using two commonly used methods.

- 1) Using String class
- 2) Using StringBufferClass

### 2.4.1. String classes

- The syntax of String class is

```
String string_variable;
```

Example:-

```
class stringDemo
{
    public static void main(String args[])
    {
        String s="Welcome to java programming";
        System.out.println(""+s);
    }
}
```

Output

Welcome to java programming

### Operation on Strings using String class.

Following are some commonly defined methods by a string class.

Method	Description
s1.charAt(position)	Return the character present at the index position.
s1.compareTo(s2)	If s1<s2 then it returns positive. If s1>s2 then it return negative and if s1=s2 then it returns zero.



---

s1.concat(s2)	It returns the concatenated string of s1 and s2.
s1.equals(s2)	If s1 and s2 are both equal then it returns true.
s1.equalsIgnoreCase(s2)	By ignoring case, if s1 and s2 are equal then it returns true.
s1.indexOf('c')	It returns the first occurrence of character 'c' in the string s1.
s1.indexOf('c', n)	It returns the position of 'c' that occur at after nth position in string s1.
s1.length()	It gives the length of string s1.
String.valueOf(var)	Converts the value of the variable passed to it into string type.

### Program for string demonstrating String class methods.

```
class StringMethodsDemo
```

```
{  
    public static void main(String[] args)  
    {  
        String s1="Java";  
        char ch;  
        //length();  
        System.out.println("The length of string "+s1+ " = " +s1.length()); //4  
        //charAt();  
        ch=s1.charAt(2);  
        System.out.println("The Char at 2 of string "+s1+ " = " +ch); //v  
        String s2="Java";  
        String s3="Programming";  
        //compareTo( )  
        System.out.println("This is for CompareTo Method");  
        System.out.println(s1.compareTo(s2)); //0  
        System.out.println(s1.compareTo(s3)); // -6  
        System.out.println(s3.compareTo(s1)); //6  
        //equals();  
        if(s1.equals(s2)==true)  
        {  
            System.out.println(s1+" and " +s2+ " are equals");  
        }  
    }  
}
```



```
//concatnation (join)
System.out.println("Concatnation of " + s1+" and " +s3+ " = "+s1.concat(s3));
}
}
```

Output:-

```
The length of string Java = 4
The Char at 2 of string Java = v
This is for CompareTo Method
0
-6
6
Java and Java are equals
Concatnation of Java and Programming = JavaProgramming
```

### 2.4.2. String Buffer

- The StringBuffer is a class which is alternative to the String class. But StringBuffer class is more flexible to use than the String class.
- That means, using StringBuffer we can insert some components to the existing string or modify the existing string but in case of String class once the string is defined then it remains fixed.
- Following are some simple methods used for StringBuffer-

Name of method	Description
append(String str)	Appends the string to the buffer
capacity()	It returns the capacity of the string buffer
charAt(int index)	It returns a specific character from the sequence which is specified by the index.
delete(int start, int end)	It deletes the characters from the string specified by the starting and ending index.
insert(int offset, char ch)	It inserts the character at the positions specified by the offset.
length()	It return the length of the string buffer.
setCharAt(int index, char ch)	The character specified by the index from the stringbuffer is set to ch.



---

setLength(int new_len)	It sets the length of the string buffer.
toString()	It converts the string representing data in this string buffer.
replace(int start, int end, String str)	It replaces the characters specified by the new string
reverse( )	The character sequence is reversed.

#### ❖ Difference between String & StringBuffer

Sr. No	String	StringBuffer
1	The length of string object is fixed.	The length of StringBuffer can be increased.
2	The String object is immutable, that means we can not modify the string once created.	The StringBuffer class is mutable.
3	It is slower in performance.	It is faster in performance.
4	It consumes more memory.	It consumes less memory.

#### Programs

- 1) Performs the following string/ string buffer operations, write java program
  - a) Accept a password from user
  - b) Check if password is correct then display “Good”, else display “Wrong”.
  - c) Display the password in reverse order
  - d) Append password with “Welcome”.

```
import java.util.*;
class StringBufferDemo
{
    public static void main(String[] args)
    {
        String s="MSBTE-S-21";
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the password: ");
        String pwd=sc.nextLine();
```



---

```
        if(s.compareTo(pwd)==0)
            System.out.println("Good");
        else
            System.out.println("Wrong");

        StringBuffer s1=new StringBuffer(pwd);
        s1=s1.reverse();
        System.out.println("The reversed string is :"+s1);

        StringBuffer s2=new StringBuffer(pwd);
        s2=s2.append(" Welcome: ");
        System.out.println(" "+s2);
    }
}
```

Output:-

```
java StringBufferDemo
Enter the password: MSBTE-S-21
Good
The reversed string is :12-S-ETBSM
MSBTE-S-21 Welcome:
```

- 2) **Write a simple java program to find the reverse string and check the weathered the entered string is palindrome or not?**

```
import java.util.*;
class ReversePalindrome
{
    public static void main(String[] args)
    {
        String s1;
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the String: ");
```





---

```
s1=sc.nextLine(); //nextLine()-- Accepting blank spaces

StringBuffer s2=new StringBuffer(s1);
s2.reverse();
System.out.println("-----");
System.out.println("Original String: "+s1);
System.out.println("Reversed String: "+s2);
System.out.println("Checking for Palindrome String: ");
System.out.println("-----");
if(s1.equals(s2.toString()))
    {
        System.out.println(s1+" is Palindrome String:");
    }
else
    System.out.println(s1+" is Not Palindrome String");
}
```

Enter the String: mom

-----

Original String: mom

Reversed String: mom

Checking for Palindrome String:

-----

mom is Palindrome String

Enter the String: Gunwant

-----

Original String: Gunwant

Reversed String: tawnuG

Checking for Palindrome String:

-----

Gunwant is Not Palindrome String



## 2.5. Vector

- The vector is a class in java that implements the dynamic array. This class stores any no. of objects of any data type.
- This class is defined by java.util package.
- In vector one can store the elements of any data type. That means in a single vector you store integer, string, double or any other data type elements altogether.
- Vector can be created like this-

```
Vector vectobj= new Vector( ); //declaring vector without size
```

```
Vector vectobj= new Vector(5); //declaring vector with size
```

### Advantages of vectors over array

- 1) Vectors contains elements of varying data types.
- 2) The size of vector can be changed whenever required.
- 3) It can store simple objects.

Following are the some most commonly used methods of vector class.

Methods	Description
void addElement(object)	For adding the element in the vector
Object elementAt(int index)	Return the element present at specified location(index) in vector.
void insertElementAt(object obj, int pos)	For inserting the element in the vector specified by its position.
boolean removeElement(object ele)	Removes the specified element
void removeAllElements()	For removing all the elements from the vector.
void removeAllElements(int pos)	The elements specified by its position gets deleted from the vector.
int capacity()	Returns the capacity of the vector.
int size()	It returns the total no. of elements present in the vector.
boolean isEmpty()	Return true if the vector is empty.
Object firstElement()	Return the first element of the vector.
int indexOf(object ele)	Returns the index of corresponding element in the vector.
void setSize(int size)	This method is for setting the size of the vector.



---

**Program 1: Write a program, to create a vector with seven elements as {10, 30, 50, 20, 40, 10, 20}. Remove element at 3<sup>rd</sup> and 4<sup>th</sup> position. Insert new element at 3<sup>rd</sup> position. Display the original and current size of the vector.**

```
import java.util.*;
class VectorDemo
{
    public static void main(String[] args)
    {
        Vector v1= new Vector(7);
        v1.addElement(10);
        v1.add(30);
        v1.add(50);
        v1.add(20);
        v1.add(40);
        v1.add(10);
        v1.add(20);
        System.out.println("The elements in the vector are:"+v1);
        System.out.println("The original size of vector : "+v1.size());
        System.out.println("Removing the Third Element ");
        v1.remove(2); //3rd element
        System.out.println("Removing the Fouth Element ");
        v1.remove(3); //4rd element
        System.out.println("Now The elements in the vector are:"+v1);

        System.out.println("Inserting element 100 at 3rd position:");
        v1.insertElementAt(100, 2); //inserting 100 at 3rd position
        System.out.println("Now The elements in the vector are:"+v1);
        System.out.println("The current size of vector : "+v1.size());
    }
}
```

Output:-



The elements in the vector are:[10, 30, 50, 20, 40, 10, 20]

The original size of vector : 7

Removing the Third Element

Removing the Fouth Element

Now The elements in the vector are:[10, 30, 20, 10, 20]

Inserting element 100 at 3rd position:

Now The elements in the vector are:[10, 30, 100, 20, 10, 20]

The current size of vector : 6

**Program 2:- Write a program to add 2 integer, 2 string and 2 float objects to a vector. Remove element specified by user and display the list.**

## 2.6. Wrapper classes

- Wrapper classes are those classes that allow primitive data types to be accessed as objects.
- The Wrapper class is one kind of Wrapper around a primitive data type.
- The wrapper classes represent the primitive data types in its instances of the class.
- Following tables shows various primitive data types and the corresponding wrapper classes.

Primitive data type	Wrapper class
boolean	Boolean
byte	Byte
char	Char
double	Double
float	Float
int	Integer
long	Long
short	Short
void	Void



### 2.6.1. Uses of Wrapper class

- Wrapper classes are used to convert numeric strings into numeric values.
- Wrapper classes are used to convert numeric values to string.
- Using the `intValue()` method we can retrieve value of the object as its primitive data type.

#### Program:-

1. Define wrapper class. Give the following wrapper class methods with syntax and use.

- 1) To convert integer number to string
- 2) To convert numeric string to integer number.

```
class WrapperDemo
{
    public static void main(String[] args)
    {
        System.out.println("Integer number to string Conversion:");
        int i=100;
        String s=Integer.toString(i);
        System.out.println("int value:"+i);
        System.out.println("Equivalent to string:"+s);
        System.out.println("-----");
        System.out.println("String to Integer Conversion:");
        String s1="200";
        int n=Integer.parseInt(s1);
        System.out.println("String value:"+s1);
        System.out.println("Equivalent to int:"+n);
    }
}
```

Output:-

Integer number to string Conversion:

int value:100

Equivalent to string:100

-----



---

String to Integer Conversion:

String value:200

Equivalent to int:200

## 2.7. Enumerated Types

- The enumerated data types can be denoted by the keyword enum.
- The enum helps to define the user defined data type.
- The value can also be assigned to the elements in the enum data type.

### Program for enum Demonstration

```
class enumDemo
{
    enum Color
    {
        Red, Green, Blue;
    }
    public static void main(String[] args)
    {
        Color c1=Color.Red;
        System.out.println(c1);
    }
}
```

### Output

Red

---

**End of Unit-II**

---

## Unit-III Inheritance, Interface and Package

### 3.1 Inheritance: -

#### Definition-

- Inheritance is a mechanism in java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.
- The inheritance can be achieved by incorporating the definition of one class into another using the keyword **extends**.

#### 3.1.1. Concept of inheritance: -

- The inheritance is a mechanism in which the child class is derived from a parent class.
- This derivation is using the keyword **extends**.

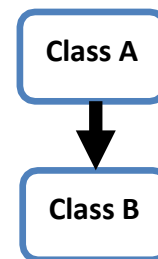
For example:

Class A <- base class

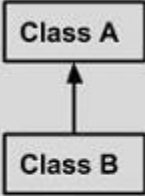
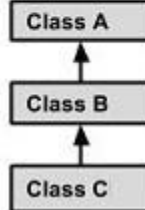
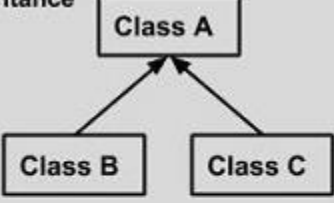
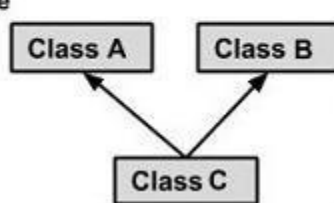
```
{  
.....  
}
```

Class B extends A

```
{  
---- //uses of properties of A  
}
```



### 3.2. Types of Inheritance

<p>Single Inheritance</p>  <pre> classDiagram     class A     class B     B -- &gt; A         </pre>	<pre> public class A {     ..... } public class B extends A {     ..... }         </pre>
<p>Multi Level Inheritance</p>  <pre> classDiagram     class A     class B     class C     B -- &gt; A     C -- &gt; B         </pre>	<pre> public class A { .....} public class B extends A { .....} public class C extends B { .....}         </pre>
<p>Hierarchical Inheritance</p>  <pre> classDiagram     class A     class B     class C     B -- &gt; A     C -- &gt; A         </pre>	<pre> public class A { .....} public class B extends A { .....} public class C extends A { .....}         </pre>
<p>Multiple Inheritance</p>  <pre> classDiagram     class A     class B     class C     C -- &gt; A     C -- &gt; B         </pre>	<pre> public class A { .....} public class B { .....} public class C extends A,B {     ..... } // Java does not support multiple Inheritance         </pre>

#### Program.1. Program for implanting Single Inheritance

```

class Teacher
{
void Display()
{
System.out.println("This is teacher ");
}
}
class student extends Teacher{
void show()
{
System.out.println("This is Student");
}
}
        
```



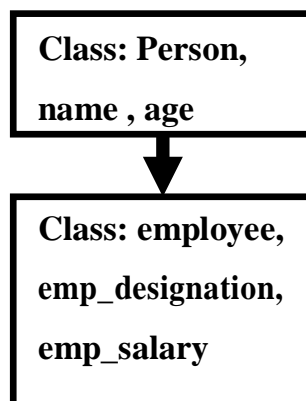


```
}  
class TestInheritance  
{  
public static void main(String args[])  
{  
Student s=new student();  
s.show();  
d.display();  
}  
}
```

### Output-

This is student  
This is teacher

**Program 2. Write a program to implement following inheritance**



```
class person  
{  
String name;  
int age;  
void accept(String n,int a)  
{  
    name=n;  
    age=a;
```



```
    }
    void display()
    {
        System.out.println("name--->" + name);
        System.out.println("age--->" + age);
    }
}

class employee extends person
{
    String emp_designation;
    float emp_salary;
    void accept_emp(String d, float s)
    {
        emp_designation = d;
        emp_salary = s;
    }

    void emp_dis()
    {
        System.out.println("emp_designation-->" + emp_designation);
        System.out.println("emp_salary-->" + emp_salary);
    }
}

class single_demo
{
    public static void main(String args[])
    {
        employee e = new employee();
        e.accept("ramesh", 35);
        e.display();
        e.accept_emp("lecturer", 35000.78f);
        e.emp_dis();
    }
}
```



```
    }  
}
```

**Program 3. Write a java program to implement multilevel inheritance with 4 levels of hierarchy.**

```
class emp  
{  
    int empid;  
    String ename;  
    emp(int id, String nm)  
    {  
        empid=id; ename=nm;  
    }  
}  
class work_profile extends emp  
{  
    String dept;  
    String job;  
    work_profile(int id, String nm, String dpt, String j1)  
    {  
        super(id,nm);  
        dept=dpt; job=j1;  
    }  
}  
class salary_details extends work_profile  
{  
    int basic_salary;  
    salary_details(int id, String nm, String dpt, String j1,int bs)  
    {  
        super(id,nm,dpt,j1);
```



---

```
        basic_salary=bs;
    }
    double calc()
    {
        double gs;
        gs=basic_salary+(basic_salary*0.4)+(basic_salary*0.1);
        return(gs);
    }
}

class salary_calc extends salary_details
{
    salary_calc(int id, String nm, String dpt, String j1,int bs)
    {
        super(id,nm,dpt,j1,bs);
    }
    public static void main(String args[])
    {
        salary_calc e1=new salary_calc(101,"abc","Sales","clerk",5000);
        double gross_salary=e1.calc();
        System.out.println("Empid :"+e1.empid);
        System.out.println("Emp name :"+e1.ename);
        System.out.println("Department :"+e1.dept);
        System.out.println("Job :"+e1.job);
        System.out.println("BAsic Salary :"+e1.basic_salary);
        System.out.println("Gross salary :"+gross_salary);
    }
}
```



---

### 3.3. Method Overloading

Def- Method Overloading means to define different methods with the same name but different parameters lists and different definitions.

It is used when objects are required to perform similar task but using different input parameters that may vary either in number or type of arguments.

Overloaded methods may have different return types.

It is a way of achieving polymorphism in java.

```
int add( int a, int b)           // prototype 1
int add( int a , int b , int c)  // prototype 2
double add( double a, double b) // prototype 3
```

#### Example :

```
class Sample
{
    int addition(int i, int j)
    {
        return i + j ;
    }
    String addition(String s1, String s2)
    {
        return s1 + s2;
    }
    double addition(double d1, double d2)
    {
        return d1 + d2;
    }
}
class AddOperation
{
    public static void main(String args[])
```



```
    {
    Sample sObj = new Sample();
    System.out.println(sObj.addition(1,2));
    System.out.println(sObj.addition("Hello ", "World"));
    System.out.println(sObj.addition(1.5,2.2));
    }
}
```

### 3.4. Constructor Overloading

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

//Java program to overload constructors in java

```
class Student
{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student(int i,String n)
    {
        id = i;
        name = n;
    }
    //creating three arg constructor
    Student(int i,String n,int a)
    {
        id = i;
        name = n;
```



```
        age=a;
    }
    void display()
    {
        System.out.println(id+ " "+name+" "+age);
    }

    public static void main(String args[])
    {
        Student s1 = new Student5(111,"Karan");
        Student s2 = new Student5(222,"Aryan",25);
        s1.display();
        s2.display();
    }
}
```

**Program 4. Define a class person with data member as AadharNo, name, Panno implement concept of constructor overloading. Accept data for 5 object and print it.**

```
import java.io.*;

class Person
{
    intAadharno;
    String name;
    String Panno;
    Person(intAadharno, String name, String Panno)
    {
        this.Aadharno = AadharNo;
        this.name = name;
        this.Panno = Panno;
    }
    Person(intAadharno, String name)
```



---

```
        {
            this.Aadharno = Aadharno;
            this.name = name;
            Panno = "Not Applicable";
        }
    void display()
    {
        System.out.println("Aadharno is :"+Aadharno);
        System.out.println("Name is: "+name);
        System.out.println("Panno is :"+Panno);
    }
    public static void main(String ar[])
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        Person p, p1, p2, p3, p4;
        int a;
        String n, pno;
        try
        {
            System.out.println("Enter Aadhar no");
            a = Integer.parseInt(br.readLine());
            System.out.println("Enter name");
            n = br.readLine();
            System.out.println("Enter panno");
            pno = br.readLine();
            p = new Person(a,n,pno);
            System.out.println("Enter Aadhar no");
            a = Integer.parseInt(br.readLine());
            System.out.println("Enter name");
            n = br.readLine();
            System.out.println("Enter panno");
            pno = br.readLine();
```





---

```
p1 = new Person(a,n,pno);
System.out.println("Enter Aadhar no");
a = Integer.parseInt(br.readLine());
System.out.println("Enter name");
n = br.readLine();
p2 = new Person(a,n);
System.out.println("Enter Aadhar no");
a = Integer.parseInt(br.readLine());
System.out.println("Enter name");
n = br.readLine();
p3 = new Person(a,n);
System.out.println("Enter Aadhar no");
a = Integer.parseInt(br.readLine());
System.out.println("Enter name");
n = br.readLine();
System.out.println("Enter panno");
pno = br.readLine();
p4 = new Person(a,n,pno);
p.display();
p1.display();
p2.display();
p3.display();
p4.display();
    }
catch(Exception e)
    {
        System.out.println("Exception caught"+e);
    }
}
```



---

### 3.5. Overriding(Method Overriding)

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java. If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding. Method overriding is used for runtime polymorphism.

Example:

```
class Vehicle
{
    void run()
    {
        System.out.println("Vehicle is running");
    }
}
class Bike extends Vehicle
{
    void run()
    {
        System.out.println("Bike is running safely");
    }
}
public static void main(String args[])
{
    Bike2 obj = new Bike2();
    obj.run();
}
```

### 3.6. Dynamic Method Dispatch (Runtime Polymorphism)

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime. This is how java implements runtime polymorphism. When an overridden method is called by a reference, java determines which version of that method to



---

execute based on the type of object it refer to. In simple words the type of object which it referred determines which version of overridden method will be called.

### Example

```
class Game
{
    public void type()
    { System.out.println("Indoor & outdoor"); }
}
```

Class Cricket extends Game

```
{
    public void type()
    {
        System.out.println("outdoor game");
    }
}
```

```
public static void main(String[] args)
{
    Game gm = new Game();
    Cricket ck = new Cricket();
    gm.type();
    ck.type();
    gm=ck; //gm refers to Cricket object
    gm.type(); //calls Cricket's version of type
}
}
```



---

**Program.5. Write a single program to implement inheritance and polymorphism in java.**

```
class Employee
{
    String name;
    String address;
    int number;
    Employee(String name, String address, int number)
    {
        System.out.println("Constructing an Employee");
        this.name = name;
        this.address = address;
        this.number = number;
    }
    public void mailCheck()
    {
        System.out.println("Mailing a check to " + this.name + " " + this.address);
    }
    public String toString()
    {
        return name + " " + address + " " + number;
    }
    public String getName()
    {
        return name;
    }
    public String getAddress()
    {
        return address;
    }
    public void setAddress(String newAddress)
```



---

```
        {
            address = newAddress;
        }
        public int getNumber()
        {
            return number;
        }
    }
}

class Salary extends Employee
{
    private double salary;
    Salary(String name, String address, int number, double salary)
    {
        super(name, address, number);
        setSalary(salary);
    }
    public void mailCheck()
    {
        System.out.println("Within mailCheck of Salary class ");
        System.out.println("Mailing check to " + getName() + " with salary " + salary);
    }
    public double getSalary()
    {
        return salary;
    }
    public void setSalary(double newSalary)
    {
        if(newSalary >= 0.0)
        {
            salary = newSalary;
        }
    }
}
```



---

```
public double computePay()
{
    System.out.println("Computing salary pay for " + getName());
    return salary/52;
}
}

public class Demo
{
    public static void main(String [] args)
    {
        Salary s = new Salary("RAM", "Dadar", 3, 3600.00);
        Employee e = new Salary("John ", "Thane", 2, 2400.00);
        System.out.println("Call mailCheck using Salary reference --");
        s.mailCheck();
        System.out.println("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}
```

### 3.7. Final Variable and Final Methods

All variable and methods can be overridden by default in subclass. In order to prevent this, the final modifier is used. Final modifier can be used with variable, method or class.

**final variable:** the value of a final variable cannot be changed.

final variable behaves like class variables and they do not take any space on individual objects of the class.

Eg of declaring final variable:

```
final int size = 100;
```



---

**final method:** making a method final ensures that the functionality defined in this method will never be altered in any way, ie a final method cannot be overridden.

Eg of declaring a final method:

```
final void findAverage()  
{  
    //implementation  
}
```

### 3.8. Use of Super Keyword

when you will want to create a superclass that keeps the details of its implementation to itself (that is, that keeps its data members private). In this case, there would be no way for a subclass to directly access or initialize these variables on its own. Whenever a subclass needs to refer to its immediate super class, it can do so by use of the keyword super.

As constructor can not be inherited, but derived class can called base class constructor using super () super has two general forms.

The first calls the super class constructor.

super() method)

The second is used to access a member of the super class that has been hidden by a member of a subclass.

#### Using super () to Call Super class Constructors

A subclass can call a constructor method defined by its super class by use of the following form of super:

```
super(parameter-list);
```

Here,

parameter-list specifies any parameters needed by the constructor in the super class. super( ) must always be the first statement executed inside a subclass" constructor.

#### A Second Use for super

The second form of super acts somewhat like this, except that it always refers to the super class of the subclass in which it is used.



---

This usage has the following general form: `super.member`

Here, `member` can be either a method or an instance variable. This second form of `super` is most applicable to situations in which member names of a subclass hide members by the same name in the super class.

**Example:** // Using `super` to overcome name hiding.

```
class A
{
    int i;
} // Create a subclass by extending class A.
class B extends A
{
    int i; // this i hides the i in A
    B(int a, int b)
    {
        super.i = a; // i in A
        i = b; // i in B
    }
    void show()
    {
        System.out.println("i in superclass: " + super.i);
        System.out.println("i in subclass: " + i);
    }
}
class UseSuper
{
    public static void main(String args[])
    {
        B subOb = new B(1, 2);
        subOb.show();
    }
}
```





---

**Q. What is importance of super and this keyword in inheritance? Illustrate with suitable example**

Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a superclass.

The class that does the inheriting is called a subclass. Therefore, a subclass is a specialized version of a superclass. Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super. Super has two general forms. The first calls the super class constructor.

The second is used to access a member of the superclass that has been hidden by a member of a subclass.

super() is used to call base class constructor in derived class. Super is used to call overridden method of base class or overridden data or evoked the overridden data in derived class.

E.g.: use of super()

```
class BoxWeight extends Box
```

```
{
```

```
    BowWeight(int a ,int b,int c ,int d)
```

```
    {
```

```
        super(a,b,c) // will call base class constructor Box(int a, int b,  
int c)
```

```
        weight=d // will assign value to derived class member weight.
```

```
    }
```

E.g.: use of super.

```
Class Box
```

```
{
```

```
    Box()
```

```
    {
```

```
    }
```

```
    void show()
```



---

```
    {
        //definition of show
    }
} //end of Box class
```

Class BoxWeight extends Box

```
{
    BoxWeight()
    {
    }
    void show() // method is overridden in derived
    {
    Super.show() // will call base class method
    }
}
```

### 3.9. Abstract Methods and Classes

Abstract methods, similar to methods within an interface, are declared without any implementation. They are declared with the purpose of having the child class provide implementation. They must be declared within an abstract class.

A class declared abstract may or may not include abstract methods. They are created with the purpose of being a super class.

Syntax

```
modifier abstract class className {
    //declare fields
    //declare methods
    abstract dataType methodName();
}

modifier class childClass extends className {
    dataType methodName(){}
}
```



---

Abstract classes and methods are declared with the 'abstract' keyword. Abstract classes can only be extended, and cannot be directly instantiated.

Abstract classes provide a little more than interfaces. Interfaces do not include fields and super class methods that get inherited, whereas abstract classes do. This means that an abstract class is more closely related to a class which extends it, than an interface is to a class that implements it.

### Example

```
public abstract class Animal
{
    String name;
    abstract String sound(); //all classes that implement Animal must have a sound method
}

public class Cat extends Animal
{
    public Cat()
    {
        this.name = "Garfield";
    }
    public String sound()
    {
        //implemented sound method from the abstract class & method
        return "Meow!";
    }
}
```

### 3.10. Static Members

Static members are data members (variables) or methods that belong to a static or a non static class itself, rather than to objects of the class. Static members always remain the same, regardless of where and how they are used. Because static members are associated with the class, it is not necessary to create an instance of that class to invoke them.

```
static data_type variable_name;
```

### Example

```
class VariableDemo
{
```



```
static int count=0;
public void increment()
{
    count++;
}
public static void main(String args[])
{
    VariableDemo obj1=new VariableDemo();
    VariableDemo obj2=new VariableDemo();
    obj1.increment();
    obj2.increment();
    System.out.println("Obj1: count is="+obj1.count);
    System.out.println("Obj2: count is="+obj2.count);
}
}
```

### Output:

```
Obj1: count is=2
Obj2: count is=2
```

## Interface:-

### ➤ Defining Interfaces:-

Interface is also known as kind of a class. Interface also contains methods and variables but with major difference, the interface consist of only abstract method (i.e. methods are not defined, these are declared only) and final fields (shared constants).

This means that interface do not specify any code to implement those methods and data fields contains only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods. An interface is defined much like class.

### Syntax:

```
access interface InterfaceName
{
    return_type method_name1(parameter list);
    ....
    return_type method_nameN(parameter list);
}
```



```
type final-variable 1 = value1;
....
type final-variable N = value n;
}
```

Where,

access is either public or not used.

‘interface’ is the java keyword and “InterfaceName” is any valid java variable.

Variables- of interface are explicitly declared final and static. This means that the implementing class cannot change them. They must be initialized with constant value.

Methods- The methods declared in interfaces are abstract, there can be no default implementation of any method specified within an interface.

Following is the example of interface definition

```
interface student
{
    static final int rollno=1;
    static final String name="Genelia"
    void show();
}
```

- **Features:**

1. Variable of an interface are explicitly declared final and static (as constant) meaning that the implementing the class cannot change them they must be initialize with a constant value all the variable are implicitly public of the interface, itself, is declared as a public
2. Method declaration contains only a list of methods without anybody statement and ends with a semicolon the method are, essentially, abstract methods there can be default implementation of any method specified within an interface each class that include an interface must implement all of the method

- **Need:**

1. To achieve multiple Inheritance.
2. We can implement more than one Interface in the one class.
3. Methods can be implemented by one or more class.

➤ **Difference between class and interface**

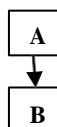
<b>Class</b>	<b>Interface</b>
It has instance variable.	It has final variable.
It has non abstract method.	It has by default abstract method.
We can create object of class.	We can,,t create object of interface.
Class has the access specifiers like public, private, and protected.	Interface has only public access specifier
Classes are always extended.	Interfaces are always implemented.
The memory is allocated for the classes.	We are not allocating the memory for the interfaces.
Multiple inheritance is not possible with classes	Interface was introduced for the concept of multiple inheritance
<pre>class Example { void method1() { Body } void method2() { body } }</pre>	<pre>interface Example { int x =5; void method1(); void method2(); }</pre>

➤ **Extending interfaces:-**

Like classes, Interfaces can also be extended. That is an interface can be sub-interfaced from other interfaces. The new sub-interfaced will inherit all the members of the sub-interfaces in the manner similar to subclasses.

- This is achieved using the keyword **extends** as shown below.

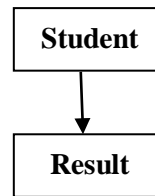
```
interface A extends B
{
    body of A.
}
```



- We can put all the constant in one interface and the methods in the other. This will enable us to use the constants in the classes where the methods are not required.

**E.g**

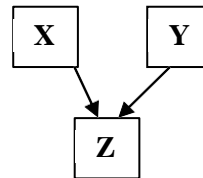
```
interface Student
{
    int RN=1;
    String name ="Genelia";
}
interface Result extends Student
{
    void display( );
}
```



- We can also combine several interfaces together into a single inheritance.

**E.g**

```
interface X
{
    int cost=100;
    String name="book";
}
interface Y
{
    void display();
}
interface Z extends X , Y
{
    -----
}
```



### ➤ **Implementing interfaces:-**

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

```
class classname implements interfacename
{
    Body of classname;
}
```

**E.g.**

```
class A implements B
{
    Body of class A;
}
```

Here, class A “implements ” the interface “B”.

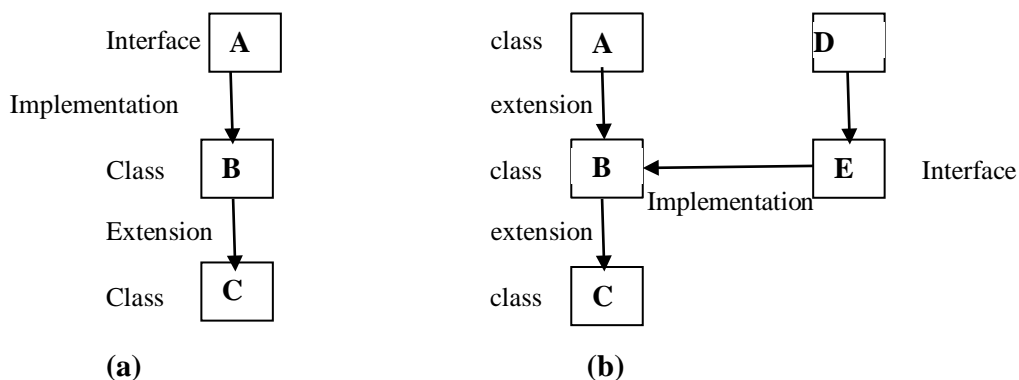
- A more general form of implementation may be as follows

```
class classname extends superclass implements interface1, interface2,.....
{
    body of classname;
}
```

**E.g.**

```
class A extends B implements X,Y, .....
{
    Body of class A;
}
```

- **Following are the various forms of interface implementation.**



### 1. Program for implementing interface:-(Use of interface )

```
interface Area // interface defined
{
    final static float pi=3.14F;
    float Cal(float x, float y);
}
class Rectangle implements Area
{
    public float Cal(float x, float y)
    {
        return(x * y);
    }
}
class Circle implements Area
{
```





```
        public float Cal(float x, float y)
        {
            return(pi *x * x);
        }
    }
```

```
class InterfaceDemo
{
    Public static void main(String args[ ])
    {
        Rectangle r = new Rectangle( );
        Circle c = new Circle();
        Area a ; // interface object
        a = r;
        System.out.println("Area of Rectangle=" +a.Cal(10,20));
        a = c;
        System.out.println("Area of Circle=" +a.Cal(10, 0));
    }
}
```

Output :-

```
C:\java\jdk1.7.0\bin> javac InterfaceDemo.java
C:\java\jdk1.7.0\bin> java InterfaceDemo
Area of Rectangle=200
Area of Circle=314
```

## 2. Program for Example of interface:- (Implementing Multilevel with hybrid inheritance)

```
interface sports
{
    int sport_wt=5;
    public void disp();
}
class Test
{
    int roll_no;
    String name;
    int m1,m2;
    Test (int r, String nm, int m11,int m12)
    {
        roll_no=r;
        name=nm;
        m1=m11;
        m2=m12;
    }
}
```



```
class Result extends Test implements sports
{
    Result (int r, String nm, int m11,int m12)
        {
            super (r,nm,m11,m12);
        }
    public void disp( )
        {
            System.out.println("Roll no : "+roll_no);
            System.out.println("Name : "+name);
            System.out.println("sub1 : "+m1);
            System.out.println("sub2 : "+m2);
            System.out.println("sport_wt : "+sport_wt);

            int t=m1+m2+sport_wt;

            System.out.println("total : "+t);
        }
}

class Demo
{
    public static void main(String args[])
    {
        Result r= new Result(101,"abc",75,75);
        r.disp();
    }
}
```

### ➤ Accessing interface:- References, Variables & Methods

Interfaces can be used to declare a set of constant that can be used in different classes. This is similar to creating header files in C++ to contain a large no. of constants. The constant values will be available to any class that implements the interface. The values can be used in declaration, or anywhere we can use a final value.

**E.g.**

```
interface A
{
    int m=10;
    int n=50;
}
class B implements A
{
    int x=m;
    void method_B(int s)
        {
            -----
        }
}
```



```
        if(s<n)
        }
    }
```

### ➤ Nesting of inheritances:-

Interfaces can be nested similar to a class. An interface can be nested within a *class* and *another interface*.

An interface can be declared inside a class body or interface body is called as nesting of interfaces. The nested interface cannot be accessed directly. To access nested interface, it must be referred by the outer interface or class.

- Rules for nesting interface:-

- 1) Nested interface must be public if it is declared inside the interface.
- 2) Nested interface can have access modifiers if it is declared inside the class.
- 3) Nested interfaces are declared as static implicitly.

- **interface inside interface-**

- **Syntax-**

```
interface interface_name
{
    ----
    interface nested_interface_name;
    {
        -----
    }
}
```

- **Nested interface within classes-**

- **Syntax-**

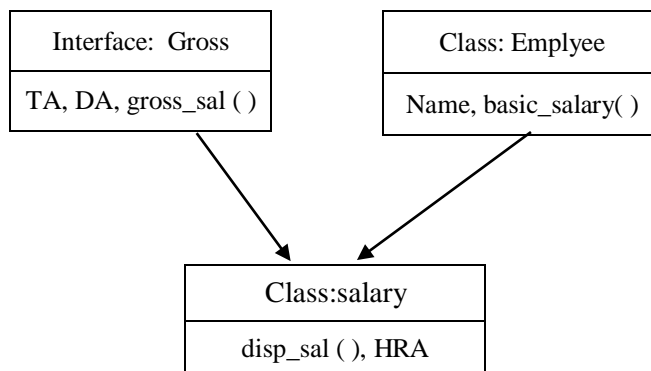
```
class class_name
{
    ----
    Interface nested_interface_name:
    {
        ----
    }
}
```

### ➤ Que. How to achieve multiple inheritance explain with suitable program?

- **Multiple inheritances:-**

It is a type of inheritance where a derived class may have more than one parent class. It is not possible in case of java as you cannot have two classes at the parent level. Instead, there can be one class and one interface at the parent level to achieve multiple interface.

Interface is similar to classes but can contain final variables and abstract methods. Interfaces can be implemented to a derived class.



### Program for above hierarchy (To achieve Multiple inheritance)

```
interface Gross
{
    double TA=800.0;
    double DA=3500;
    void gross_sal();
}
class Employee
{
    String name;
    double basic_sal;

    Employee(String n, double b)
    {
        name=n; basic_sal=b;
    }
    void display()
    {
        System.out.println("Name of Employee :"+name);
        System.out.println("Basic Salary of Employee :"+basic_sal);
    }
}

class Salary extends Employee implements Gross
{
    double HRA;
    Salary(String n, double b, double h)
    {
        super(n,b);
        HRA=h;
    }
    void disp_sal()
    {
        display();
        System.out.println("HRA of Employee :"+hra);
    }
}
```

```

        public void gross_sal()
        {
            double gross_sal=basic_sal + TA + DA + HRA;
            System.out.println("Gross salary of Employee :"+gross_sal);
        }
    }
class EmpDetails
{
    public static void main(String args[])
    {
        Salary s=new Salary("Sachin",8000,3000);
        s.disp_sal();
        s.gross_sal();
    }
}
    
```

### 3.2 Packages-

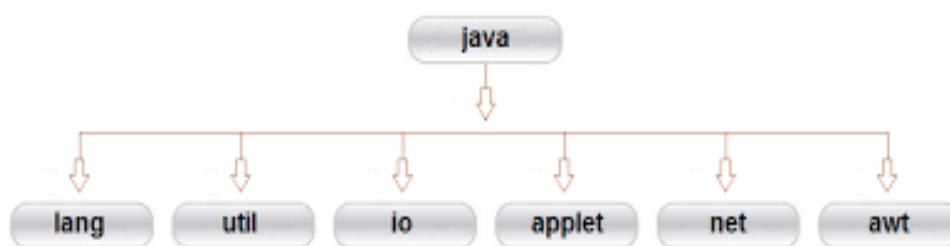
Java provides a mechanism for partitioning the class namespace into more manageable parts called package (i.e package are container for a classes). The package is both naming and visibility controlled mechanism.

We can define classes inside a package that are not accessible by code outside that package. We can also define class members that are only exposed to members of the same package.

#### ➤ Java API packages:-

The java API provides a no. of classes grouped into different packages according to their functionality.

The following fig1. shows the java API packages.



*fig1. Java API Packages*

- Packages and their use

Package Name	Description (Use of Package)
java.lang	Language support classes. These are classes that java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions.



java.util	Language utility classes such as vectors, hash tables, random numbers, date etc.
java.io	Input/output support classes. They provide facilities for the input and output of data
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.applet	Classes for creating and implementing applets.

- **Naming Conventions:-**

Packages in java can be named using standard java naming rules.

**E.g.**

1) java.awt.Color;

**awt-** package name, **Color-**class name

2) double x= java.lang.Math.sqrt(a);

**lang-** package name, **math-** class name, **sqrt-** method name

➤ **Creating Packages:- (Defining Packages)**

Creation of packages includes following steps.

1) Declare a package at the beginning of the file using the following form.

**package** *package\_name*

**e.g.**

package pkg; - name of package

package is the java keyword with the name of package. This must be the first statement in java source file.

2) Define a class which is to be put in the package and declare it public like following way.

```
Package first_package;  
Public class first_class  
{  
----  
Body of class;  
}
```

In above example, “first\_package” is the package name. The class “first\_class” is now considered as a part of this package.



- 3) Create a sub-directory under the directory where the main source files are stored.
- 4) Store the listing of, as classname.java file is the sub-directory created.

e.g. from the above package we can write “first\_class.java”

- 5) Compile the source file. This creates .class file is the sub-directory. The .class file must be located in the package and this directory should be a sub-directory where classes that will import the package are located.

➤ **Que. How to create package?**

The syntax for creating package is:

**package *pkg*;**

Here, *pkg* is the name of the package

e.g : package mypack;

Packages are mirrored by directories. Java uses file system directories to store packages. The class files of any classes which are declared in a package must be stored in a directory which has same name as package name. The directory must match with the package name exactly.

A hierarchy can be created by separating package name and sub package name by a period(.) as pkg1.pkg2.pkg3; which requires a directory structure as pkg1\pkg2\pkg3. The classes and methods of a package must be public.

➤ **Accessing a package:-**

To **access** package In a Java source file, **import** statements occur immediately. Following the **package** statement (if it exists) and before any class definitions.

**Syntax:**

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

Here, “pkg1” is the name of the top level package. “pkg2” is the name of package is inside the package1 and so on.

“classname”-is explicitly specified statement ends with semicolon.

- **Second way** to access the package

**import packagename.\*;**



Here, packagename indicates a single package or hierarchy of packages.  
The star(\*) denotes the compiler should search this entire package hierarchy when it encountered a class.

- **Access Specifiers /Access Modifiers in Package**

Access Modifier → Access Location ↓	Public	Private	Protected
Same class	Yes	Yes	Yes
Subclass in same package	Yes	No	Yes
Other classes in same package	Yes	No	Yes
Subclass in other packages	Yes	No	Yes
Non subclasses in other packages	Yes	No	No

➤ **Example of creating package and accessing the package.**

**package1:**

```
package package1;
public class Box
{
    int l= 5;
    int b = 7;
    int h = 8;
    public void display()
    {
        System.out.println("Volume is:"+(l*b*h));
    }
}
```

**Source file:**

```
import package1.Box;
class VolumeDemo
{
    public static void main(String args[])
    {
        Box b=new Box();
        b.display();
    }
}
```





- **Que:- Design a package containing a class which defines a method to find area of rectangle. Import it in java application to calculate area of a rectangle.**

```
package Area;
public class Rectangle
{
    doublelength,bredth;
    public doublerect(float l,float b)
    {
        length=l;
        bredth=b;
        return(length*bredth);
    }
}

import Area.Rectangle;

class RectArea
{
    public static void main(String args[])
    {
        Rectangle r=new Rectangle( );
        double area=r.rect(10,5);
        System.out.println("Area of rectangle= "+area);
    }
}
```

- **Adding class to a package:-**

Consider the we have package 'P' and suppose class B is to be added to following package.

```
Package P;
{
    Public class X;
    {
        //body of X;
    }
}
```

The package 'P' contains one public class named as 'A'  
For adding class X to this package follows given steps

- 1) Define the class & make it as public.
- 2) Place package statement.



---

Package P1;  
Before the class definition as follows

```
Package P1;  
    Public class B  
    {  
        // Body of B  
    }
```

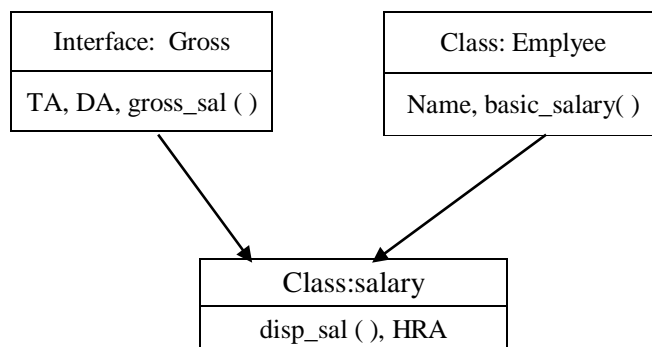
- 3) Store this as B.java file under directory P1;
- 4) Compile B.java file. This will create B.class file and place it in the directory P1.  
Now, the package “P2” contains both the classes A and B. So to import both classes use **import P1.\***

➤ **Very Important Questions:-**

- 1) Explain the types of Inheritance?
- 2) Write a java program to implements to implements multilevel inheritance with 4 levels of hiererachy?
- 3) What is the single level inheritance? Explain with suitable example?
- 4) Explain inheritance and polymorphism features of java?
- 5) Explain constructor overloading?
- 6) Explain method overriding?
- 7) Explain final variables and final methods.?
- 8) Describe the final method and final variables with respect to inheritance?
- 9) Explain abstract methods and classes?
- 10) Explain static members?
- 11) What is meant by an interface? State its need and write syntax and features of an interface. Give one example
- 12) State any four system packages along with their use.
- 13) Write the effect of access specifiers public, private and protected in package.
- 14) Explain with example how to achieve multiple inheritance with interface.
- 15) What is package ? State any four system packages along with their use ? How to add class to a user defined package ?
- 16) Write syntax of defining interface. Write any major two differences between interface and a class.
- 17) Design a package containing a class which defines a method to find area of rectangle. Import it in Java application to calculate area of a rectangle.



- 
- 18) What is interface ? How it is different from class ? With suitable program explain the use of interface.
- 19) What is package ? How to create package ? Explain with suitable example.
- 20) Write a program to implement following hierarchy





## Unit – IV:

### 1. Types of errors, exceptions, try...catch statement, multiple catch blocks, throw and throws keywords, finally clause, uses of exceptions, user defined exceptions

#### Types of errors

Introduction Errors and Exception:

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.
- Error may produce
  - An incorrect output
  - may terminate the execution of program abruptly
  - may cause the system to crash
- It is important to detect and manage properly all the possible error conditions in program.

#### Types of Errors

1. Compile time Errors: Detected by javac at the compile time
2. Run time Errors: Detected by java at run time
3. Logical Errors: Produces incorrect results.

#### 1. Compile Time Errors(Syntactical errors)

- Errors which are detected by javac at the compilation time of program are known as compile time errors.
- Most of compile time errors are due to typing mistakes, which are detected and displayed by javac.
- Whenever compiler displays an error, it will not create the .class file.
- Typographical errors are hard to find.

The most common problems:

- Missing semicolon
- Missing or mismatch of brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in strings
- Use of undeclared variables
- Incompatible types in assignments/ Initialization
- Bad references to objects
- Use of = in place of == operator etc.
- Other errors are related to directory path like command not found

#### Example:

```
class ExError
```

```
{  
    public static void main(String args[])  
    {  
        System.out.println("Preeti Gajjar") //Missing semicolon  
    }  
}
```



```
}  
}
```

### Output:

Javac detects an error and display it as follow: Error1.java: 5: ,,;  
,,expected System.out.println (“Preeti Gajjar”)  
^1 error

## 2. Run time Error(Logical Error)

- There is a time when a program may compile successfully and creates a .class file but may not run properly.
- It may produce wrong results due to wrong logic or may terminate due to errors like stack overflow, such logical errors are known as run time errors.
- Java typically generates an error message and aborts the program.
- It is detected by java (interpreter)
- Run time error causes termination of execution of the program.
- The most common run time errors are:
  - Dividing an integer by zero.
  - Accessing element that is out of the bounds of an array.
  - Trying to store a value into an array of an incompatible class or type.
  - Passing a parameter that is not in a valid range or value for a method.
  - Attempting to use a negative size of an array
  - Converting invalid string to a number.

### Example:

```
class Error2  
{  
    public static void main(String args[])  
    {  
        int a=10,b=5 ,c=5;  
        int x = a / (b-c);          // division by zero  
        System.out.println(“x=“ + x); int y = a / (b+c);  
        System.out.println(“y=“ + y);  
    }  
}
```

## 3. Semantic Errors(Logical Error)

- They are known as Logical errors, they occurs when the code runs without errors, but produces incorrect results.
- Logical errors are caused by mistakes in design or implementation of code.
- It is difficult to find and fix.

### Example:

```
class Error3  
{  
    public static void main(String args[])  
    {  
        int a=10,b=5  
        int c = a + b;  
        c = a - b; // Incorrect calculation  
    }  
}
```

```
        System.out.println("Result="+c);
    }
}
```

**Output:**

Result= 5 instead of 15

➤ **Exception Hierarchy**

- All exception and error types are subclasses of class **Throwable**, which is the base class of the hierarchy.
- Immediately below Throwable, are two subclasses that partition exceptions into two distinct branches.
  - Exception Class :
    - One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch.
    - Ex.NullPointerException is an example of such an exception.
  - Error Class
    - Another branch, **Error** is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE).
    - Ex.StackOverflowError is an example of such an error.

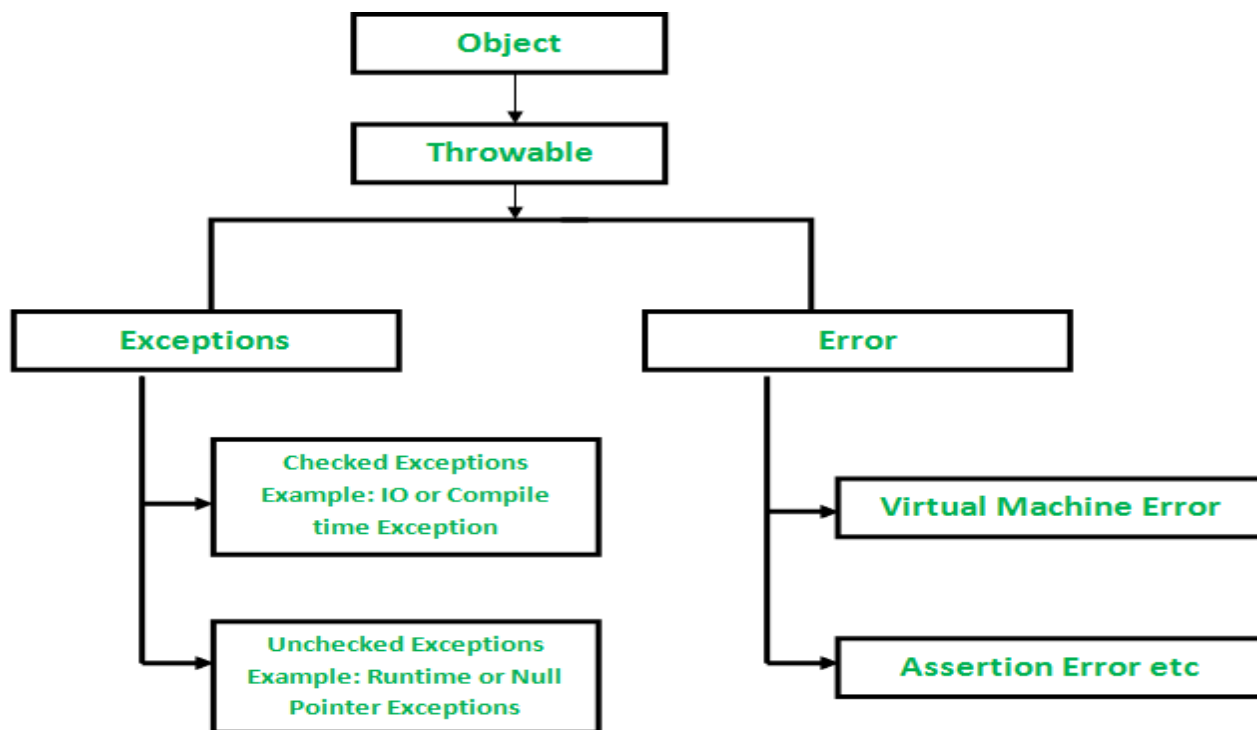
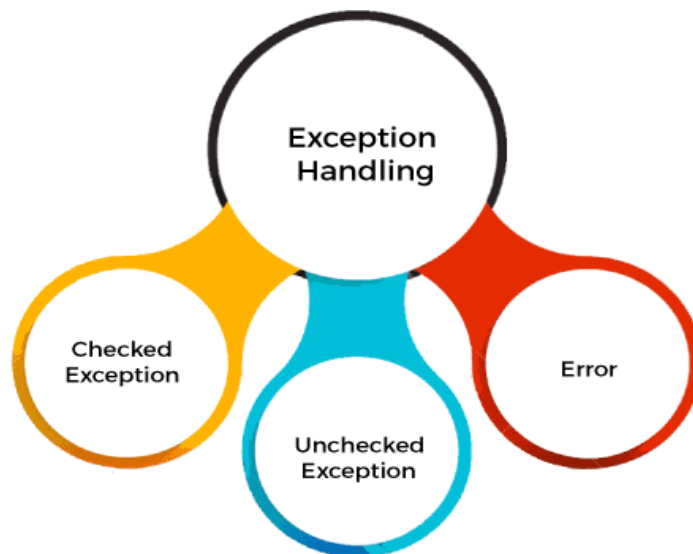


Figure: Exception Hierarchy

## ➤ Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. **Checked Exception**
2. **Unchecked Exception**
3. **Error**



Difference between Checked and Unchecked Exceptions

### 1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

## ➤ Exception Handling Mechanism:

In Java exception handling is done using **five keywords**:

1. try
2. catch
3. throw
4. throws
5. finally

## • Exceptions, try...catch statement

- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The try and catch keywords come in pairs:

```
try {
    // Block of code to try
}
catch(Exception e) {
    // Block of code to handle errors
}
```

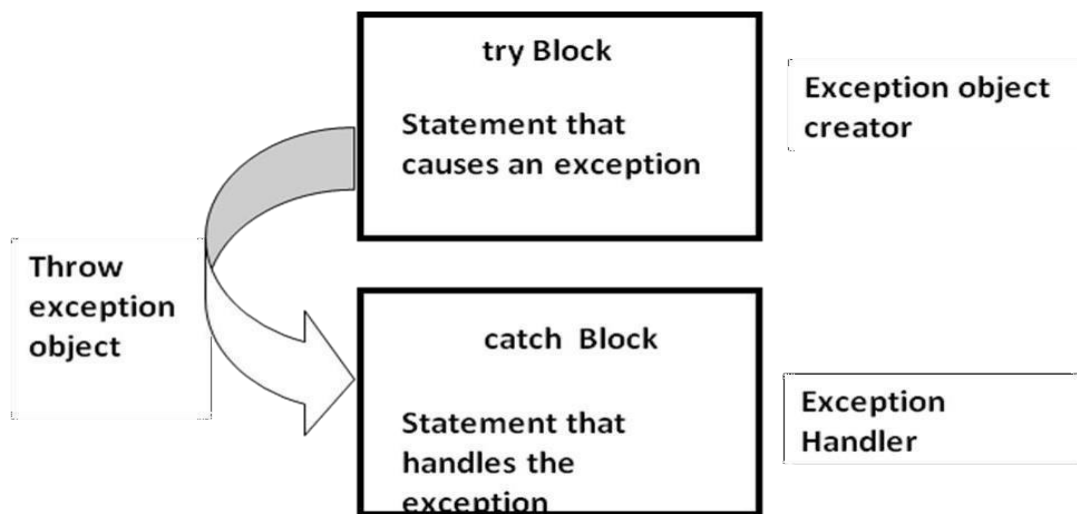


Fig. : Exception handling mechanism

### Example:

```
class TryCatchEx
{
    public static void main(String args[])
    {
        try
        {
            int a=10,b=0;
            int x = a / b;    // Arithmetic exception
            System.out.println("Result=" +c);
        }
        catch(ArithmeticException e)
        {
            System.out.println("Caught an exception:" + e.getMessage());
        }
        System.out.println("End of the program"); //This line will execute
    }
}
```



```
    }  
}
```

### Output:

```
Caught an exception: / by Zero  
End of the program
```

### • Multiple catch blocks

- A try block can be followed by one or more catch blocks.
- Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.
- Syntax:

```
try  
{  
    //Statements  
}  
catch(Exception e1)  
{  
    ...  
}  
catch(Exception e2)  
{  
    ...  
}
```

- Example:

```
public class MultipleCatchBlock1 {  
    public static void main(String[] args) {  
        try{  
            int a[]=new int[5];  
            a[5]=30/0;  
        }  
        catch(ArithmeticException e)  
        {  
            System.out.println("Arithmetic Exception occurs");  
        }  
        catch(ArrayIndexOutOfBoundsException e)  
        {  
            System.out.println("ArrayIndexOutOfBoundsException occurs");  
        }  
        catch(Exception e)  
        {
```



```
        System.out.println("Parent Exception occurs");
    }
    System.out.println("rest of the code");
}
}
```

**Output:**

Arithmetic Exception occurs  
rest of the code

- **throw and throws keywords**

- **throw keyword:**

- We can make a program to throw an exception explicitly using throw statement.
- throw throwableInstance;
- throw keyword throws a new exception.
- An object of class that extends throwable can be thrown and caught.
- Throwable ->Exception -> MyException
- The flow of execution stops immediately after the throw statement ;
- Any subsequent statements are not executed. the nearest enclosing try block is inspected to see if it has catch statement that matches the type of exception.
- If it matches, control is transferred to that statement
- If it doesn't match, then the next enclosing try statement is inspected and so on. If no catch block is matched than default exception handler halts the program.
- Syntax:

```
throw new throwable_instance;
```

- Example:

```
throw new ArithmeticException ();
throw new MyException();
```

Here ,new is used to construct an instance of MyException().

- ✓ All java's runtime exceptions have at least two constructors:
  - 1) One with no parameter
  - 2) One that takes a string parameter.
- ✓ In that the argument specifies a string that describe the exception. this string is displayed when object as used as an argument to print() or println() .
- ✓ It can also obtained by a call to getMessage(),a method defined by Throwable class.

**Example:**

Throw an exception if **age** is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```
public class Main {
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Access denied - You must be at least 18 years old.");
        }
    }
}
```

```
    }
    else {
        System.out.println("Access granted - You are old enough!");
    }
}
public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
}
}
```

**Output:**

Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.

```
    at Main.checkAge(Main.java:4)
    at Main.main(Main.java:12)
```

**➤ throws keywords**

- The Java throws keyword is used to declare an exception.
- It gives an information to the programmer that there may occur an exception.
- So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.
- Exception Handling is mainly used to handle the checked exceptions.
- If there occurs any unchecked exception such as NullPointerException, it is programmers' fault that he is not checking the code before it being used.
- Syntax:

```
    return_type method_name() throws exception_class_name{
        //method code
    }
```

- **Example:**

```
import java.io.IOException;
class Testthrows1{
    void m() throws IOException{
        throw new IOException("device error");//checked exception
    }
    void n() throws IOException{
        m();
    }
    void p(){
        try{
            n();
        }catch(Exception e){System.out.println("exception handled");}
    }

    public static void main(String args[]){
        Testthrows1 obj=new Testthrows1();
        obj.p();
        System.out.println("normal flow...");
    }
}
```

```

}
}

```

**Output:**

```

exception handled
normal flow...

```

- **finally clause**

- It can be used to handle an exception that is not caught by any of the previous catch statements.
- It can be used to handle any exception generated within a try block.
- It may be added immediately after try block or after the last catch block.
- finally block is guaranteed to execute if no any exception is thrown.
- **Use:**
  1. closing file , record set
  2. closing the connection with database
  3. releasing system resources
  4. releasing fonts
  5. Terminating network connections, printer connection etc.
  - 6.

We can use finally in this two way.

**Syntax:1**

```

try
{
.....
}
finally
{
.....
}

```

**Syntax:2**

```

try
{
.....
}
catch(.....)
{
.....
}

catch (.....)
{
.....
}
finally
{

```

**Example:**

```

    }
    .....
}
class finallyEx {
    public static void main(String[] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        }
        catch (Exception e) {
            System.out.println("Something went wrong.");
        }
        finally {
            System.out.println("The 'try catch' is finished.");
        }
    }
}

```

**Output:**

Something went wrong.  
The 'try catch' is finished.

- **Uses of exceptions**

The *advantages of Exception Handling in Java* are as follows:

1. Provision to Complete Program Execution
2. Easy Identification of Program Code and Error-Handling Code
3. Propagation of Errors
4. Meaningful Error Reporting
5. Identifying Error Types

- **User defined exceptions / Custom exception**

- In java, we can create our own exceptions by extending Exception class
- User defined exceptions are known as Custom exception.
- It is useful when we want to create specific exception for our program.
- You can define constructor for your Exception subclass or you can override toString() function to display customized message on catch.
- **reasons to use custom exceptions:**
  - To catch and provide specific treatment
  - Business logic exceptions

**Example:**

```

class std extends Exception
{
    private int rollno;
    std(int a)

```

```
{
    rollno = a ;
}
public String toString()
{
    return "MyException[" + rollno + "] is less than zero";
}
}
class Student
{
    Static void sum(int a, int b) throws std
    {
        if(a<0)
        {
            throw new std(a); //throws user defined exception object
        }
        else
        {
            System.out.println(a+b);
        }
    }
    public static void main(String[] args)
    {
        try
        {
            sum(-10,10);
        }
        catch(std me)
        {
            System.out.println(me);
        }
    }
}
```

**Output:**

MyException[" -10 "] is less than zero

- **Common java Exceptions**

SR NO.	Exception Type	Cause of Exception
1	ArithmeticException	Caused by math error such as divide by zero
2	ArrayIndexOutOfBoundsException	Caused by bad array indexes
3	ArrayStoreException	Caused when a program tries to store the wrong type of data in an array
4	FileNotFoundException	Caused by an attempt to access a nonexistent file
5	IOException	Caused by general I/O failures, such as inability to read from a file
6	NullPointerException	Caused by referencing a null object
7	NumberFormatException	Caused when a conversion between strings and number fails
8	OutOfMemoryException	Caused when there's not enough memory to allocate a new object
9	SecurityException	Caused when an applet tries to perform an action not allowed by the browser's security

## 2. Concept of Multithreading, Creating thread, extending Thread class, implementing Runnable interface, life cycle of a thread, Thread priority, Thread exception handing in threads

### ➤ Concept of Multithreading

- **Multithreading** is a conceptual programming paradigm where a program (or process) is divided into two or more subprograms(or process),which can be implemented at the same time in parallel.
- Threads in java are subprograms of main application program and share the same memory space.
- Multithreading is similar to dividing a task into subtasks and assigning them to different people for execution independently and simultaneously, to achieve a single desire.
- We use multithreading than multiprocessing because threads use a shared memory area.
- It is mostly used in games, animation, etc.

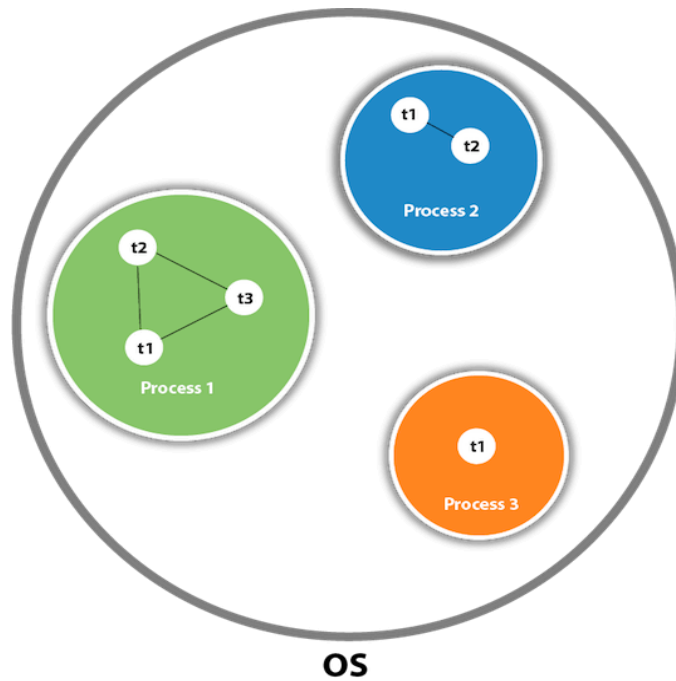
### Advantages of Java Multithreading

- 1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
- 2) You **can perform many operations together, so it saves time.**

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

▪ **Concept of thread**

- Definition: “A thread is a lightweight subprocess, the smallest unit of processing.”
- It is a separate path of execution.
- Threads are independent.
- If exception occurs in one thread, it doesn't affect other threads. It uses a shared memory area.
- We can increase the speed of application using thread.
- Thread is executed inside the process as shown in figure below:



- A thread is executed along with its several operations within a single process as shown in figure below:

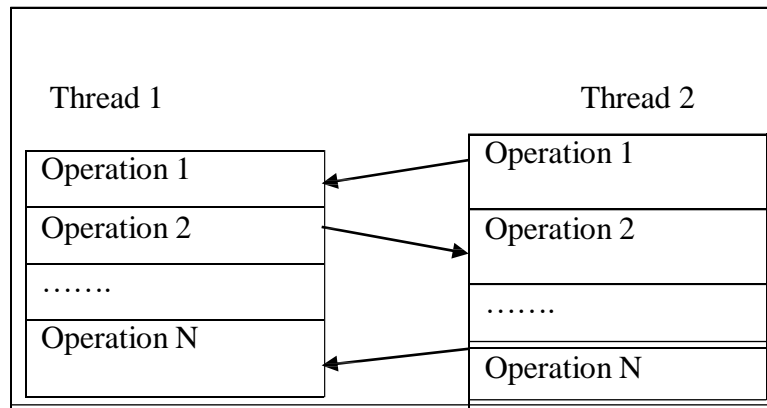
Operation 1
Operation 2
.....
Operation N

**Single Process  
Thread 1**

- A multithreaded programs contain two or more threads that can run concurrently.
- Threads share the same data and code.



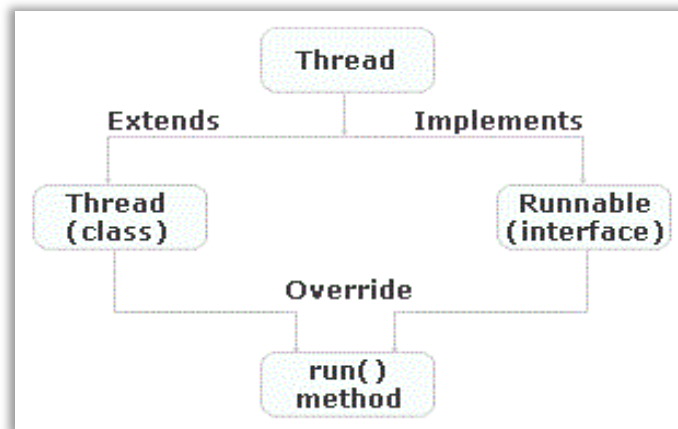
## MultiThreading Concept



### ➤ Creating thread

Thread can be implemented through any one of two ways:

1. Extending the Java.lang.Thread class
2. Implementing Java.lang.Runnable interface



To execute a thread, it is first created and then start() method is invoked on the thread. Then thread would execute and run method would be invoked.

### • Extending Thread class

1. Extending the java.lang.Thread class:
  - a. Extend Java.lang.Thread class
  - b. Override run() method in subclass from Thread class
  - c. Create instance of this subclass. This subclass may call a Thread class constructor by subclass constructor.
  - d. Invoke start() method on instance of class to make thread eligible for running.

**Commonly used methods of Thread class:**

getName()	Returns the name of the thread.
setName()	Changes the name of the thread
run()	Used to perform action for a thread
sleep()	Suspend a thread for period of time
isAlive()	Test if the thread is alive.
join()	Wait for a thread to die.
getPriority()	Returns the priority of the thread.
setPriority()	Changes the priority of the thread.
suspend()	Used to suspend the thread.
resume()	Used to resume the suspended thread.
stop()	Used to stop the thread.

**Example:**

```
class Multi extends Thread
{
    public void run(){
        System.out.println("thread is running...");
    }
    public static void main(String args[]){
        Multi t1=new Multi();
        t1.start();
    }
}
```

**Output:**

thread is running...

- **Implementing Runnable interface**

2. Implementing Java.lang.Runnable interface

- a. An object of this class is Runnable object.
- b. Create an object of Thread class by passing a Runnable object as argument.
- c. Invoke start() method on the instance of Thread class.

**Example:**

```
class Multi3 implements Runnable
{
    public void run(){
        System.out.println("thread is running...");
    }
    public static void main(String args[]){
        Multi3 m1=new Multi3();
        Thread t1 =new Thread(m1); // Using the constructor Thread(Runnable r)
        t1.start();
    }
}
```

**Output:**

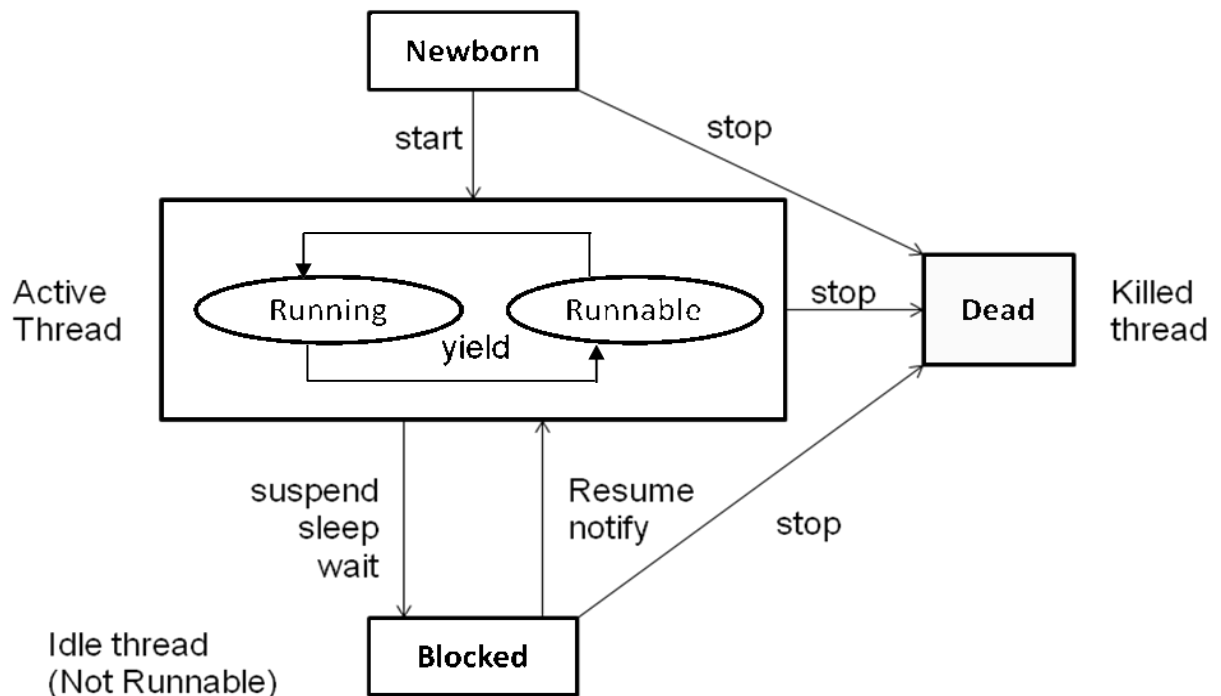
thread is running...

## ➤ Life cycle of a thread

During the life time of a thread ,there are many states it can enter, They are:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

A thread is always in one of these five states. It can move from one state to another via a variety of ways.



**Figure: Thread Life cycle Or Thread state transition Diagram**

### 1) New born state

- Whenever a new thread is created, it is always in the new state.
- At this time we can scheduled it for running, using start() method or kill it using stop () method.
- If we attempt to use any other method at this stage, an exception will be thrown.

### 2) Runnable state

- The thread is ready for execution and is waiting for the availability of the processor.
- The threads has joined waiting queue for execution.
- If all threads have equal priority, then they are given time slots for execution in roundrobin fashion. i.e. first-come, first serve manner.
- This process of assigning time to thread is known as **time-slicing**.
- If we want a thread to relinquish(leave) control to another thread of equal priority before its turn comes, then yield( ) method is used.

### 3) Running state

- The processor has given its time to the thread for its execution.
- The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.
- A running thread may change its state to another state in one of the following situations.
  - 1) When It has been suspended using suspend( ) method.
  - 2) It has been made to sleep( ).
  - 3) When it has been told to wait until some events occurs.

### 4) Blocked state/ Waiting

- A thread is waiting for another thread to perform a task. The thread is still alive.
- A blocked thread is considered “not runnable” but not dead and so fully qualified to run again.

### 5) Dead state/ Terminated

- Every thread has a life cycle. A running thread ends its life when it has completed executing its run ( ) method.
- It is natural death. However we can kill it by sending the stop message.
- A thread can be killed as soon it is born or while it is running or even when it is in “blocked” condition.

### Thread class's methods:

Thread class defines several methods that help manage threads.

No.	Method	Meaning
1	getName( )	Obtain a thread's name.
2	setName( )	Set a thread's name.
3	getPriority( )	Obtain a thread's priority
4	setPriority( )	Set a thread's priority
5	isAlive( )	Determine if a thread is still running.
6	join( )	Wait for a thread to terminate.
7	run( )	Entry point for the thread.
8	sleep( )	Suspend a thread for a period of time.
9	start( )	Start a thread by calling run method

## ➤ Thread priority

- Every thread in java has its own priority.
- Thread priority are used by thread scheduler to decide when each thread should be allowed to run.
- In Java, thread priority ranges between:
  - MIN-PRIORITY( a contant of 1 )
  - MAX-PRIORITY( a contant of 10 )
  - NORM-PRIORITY (a contant of 5) , it is default.
- **Example:**

```

class display implements Runnable
{
    public void run()
    {
        int i= 0;
        while(i < 3)
            System.out.println ("Hello:" + i++);
    }
}
public class Student
{
    public static void main(String args[])
    {
        display d = new display();
        Thread t1 = new Thread(d);
        Thread t2 = new Thread(d);
        System.out.println("Current priority of thread t1 is:" + t1.getPriority());
        t1.setpriority(3);
        t1.start();
        t2.start();
        System.out.println("New priority of thread t1 is:" + t1.getPriority());
    }
}

```

### Output:

```

F:\PG>javac Student.java
F:\PG>java Student
Current priority of thread t1 is:5
New priority of thread t1 is:3
Hello:0
Hello:1
Hello:2
Hello:0
Hello:1
Hello:2

```

## ➤ Thread exception handing in threads

- When we create a thread start() method performs some internal process and then calls run() method.
- The start() method does not start another thread of control but run() is not really “main” method of new thread.
- All uncaught exceptions are handled by code outside the run() method before the thread terminates.
- It is possible for a program to write a new default exception handler.
- The default exception handler is the uncaughtException method of the Thread group class.
- Whenever uncaught exception occurs in thread’s run method, we get a default exception dump which gets printed on System.err stream.

Thread class has two methods:

1. setDefaultUncaughtExceptionHandler()
2. setUncaughtExceptionHandler()

### Example:

```
class MyThread extends Thread{
    public void run(){
        System.out.println("Throwing in " +"MyThread");
        throw new RuntimeException();
    }
}
public class Main {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
        try {
            Thread.sleep(1000);
        } catch (Exception x) {
            System.out.println("Caught it" + x);
        }
        System.out.println("Exiting main");
    }
}
```

### Output:

```
Throwing in MyThread
Exception in thread "Thread-0" java.lang.RuntimeException
    at testapp.MyThread.run(Main.java:19)
Exiting main
```



---

## **Topic 5**

# **Java Applets & Graphics Programming**

*Total Marks- 20*

### **Specific Objectives:**

**The students will be able to write interactive applets and make use of graphics in programming.**

**They will also learn to change the background and the foreground color and to use the different fonts.**

### **Contents:**

5.1 Introduction to applets Applet, Applet life cycle (skeleton), Applet tag, Adding Applet To HTML file, passing parameter to applet, embedding <applet>tags in java code, adding controls to applets.

5.2 Graphics Programming Graphics classes, lines, rectangles, ellipse, circle, arcs, polygons, color & fonts, setColor(), getColor(), setForeground(), setBackground(), font class, variable defined by font class: name, pointSize, size, style, font methods: getFamily(), getFont(), getFontname(), getSize(), getStyle(), getAllFonts() & getavailablefontfamilyname() of the graphics environment class.



---

## Topic 5 Java Applets & Graphics Programming

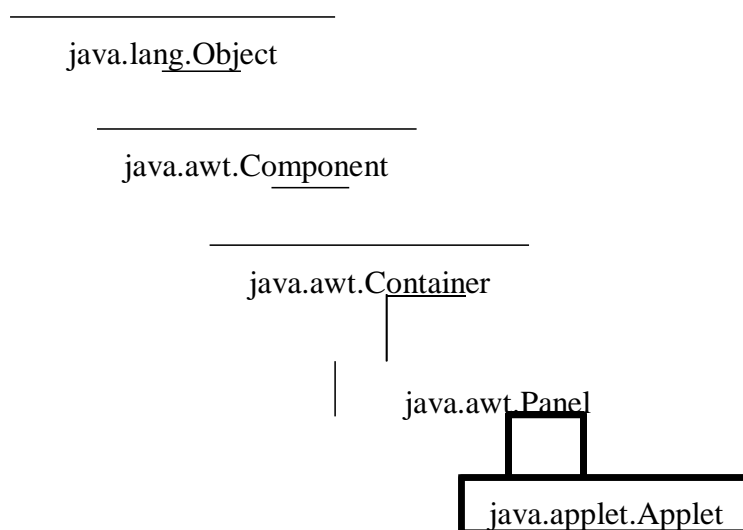
### 5.1.Introduction to applets

#### Applet Basics-

Basically, an applet is dynamic and interactive java program that inside the web page or applets are small java programs that are primarily used in internet computing. The java application programs run on command prompt using java interpreter whereas the java applets can be transported over the internet from one computer to another and run using the appletviewer or any web browser that supports java.

An applet is like application program which can perform arithmetic operations, display graphics, play sounds accept user input, create animation and play interactive games. To run an applet, it must be included in HTML tags for web page. Web browser is a program to view web page.

Every applet is implemented by creating sub class of Applet class. Following diagram shows the inheritance hierarchy of Applet class.



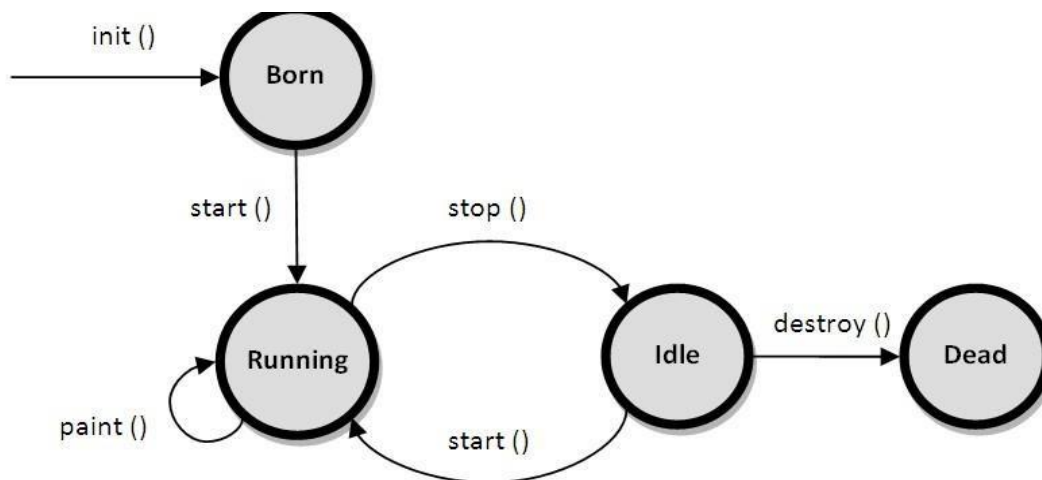
*Fig. Chain of classes inherited by Applet class in java*

#### 5.1.1 Differentiate between applet and application (4 points). [W-14, S-15, W-15 ]

Applet	Application
Applet does not use main() method for initiating execution of code	Application use main() method for initiating execution of code
Applet cannot run independently	Application can run independently
Applet cannot read from or write to files in local computer	Application can read from or write to files in local computer
Applet cannot communicate with other servers on network	Application can communicate with other servers on network
Applet cannot run any program from local computer.	Application can run any program from local computer.
Applet are restricted from using libraries from other language such as C or C++	Application are not restricted from using libraries from other language



### 5.1.2 Applet life Cycle [ W-14, S-15 ]



Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document.

The applet states include:

**Born or initialization state**

**Running state**

**Idle state**

**Dead or destroyed state**

**a) Born or initialization state [S-15]**

Applet enters the initialization state when it is first loaded. This is done by calling the `init()` method of Applet class. At this stage the following can be done:

Create objects needed by the applet

Set up initial values

Load images or fonts

Set up colors

Initialization happens only once in the life time of an applet.

```

public void init()
{
    //implementation
}
  
```

**b) Running state: [S-15]**

Applet enters the running state when the system calls the `start()` method of Applet class. This occurs automatically after the applet is initialized. `start()` can also be called if the applet is already in idle state. `start()` may be called more than once. `start()` method may be overridden to create a thread to control the applet.

```

public void start()
{
    //implementation
}
  
```



**c) Idle or stopped state:**

An applet becomes idle when it is stopped from running. Stopping occurs automatically when the user leaves the page containing the currently running applet. `stop()` method may be overridden to terminate the thread used to run the applet.

```
public void stop()
{
    //implementation
}
```

**d) Dead state:**

An applet is dead when it is removed from memory. This occurs automatically by invoking the `destroy` method when we quit the browser. Destroying stage occurs only once in the lifetime of an applet. `destroy()` method may be overridden to clean up resources like threads.

```
public void destroy()
{
    //implementation
}
```

**e) Display state: [S-15]**

Applet is in the display state when it has to perform some output operations on the screen. This happens after the applet enters the running state. `paint()` method is called for this. If anything is to be displayed the `paint()` method is to be overridden.

```
public void paint(Graphics g)
{
    //implementation
}
```

### 5.1.3 Applet Tag & Attributes [ W-15, S-16 ]

**APPLET Tag:**

The `APPLET` tag is used to start an applet from both an HTML document and from an applet viewer.

**The syntax for the standard `APPLET` tag:**

`<APPLET`

```
[CODEBASE = codebaseURL]
CODE = appletFile
[ALT = alternateText]
[NAME = appletInstanceName]
WIDTH = pixels HEIGHT = pixels
[ALIGN = alignment]
[VSPACE = pixels] [HSPACE = pixels]>
[< PARAM NAME = AttributeName1 VALUE = AttributeValue>]
[< PARAM NAME = AttributeName2 VALUE = AttributeValue>]
...
```

`</APPLET>`



---

**CODEBASE** is an optional attribute that specifies the base URL of the applet code or the directory that will be searched for the applet's executable class file.

**CODE** is a required attribute that give the name of the file containing your applet's compiled class file which will be run by web browser or appletviewer.

**ALT:** Alternate Text. The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser cannot run java applets.

**NAME** is an optional attribute used to specifies a name for the applet instance.

**WIDTH AND HEIGHT** are required attributes that give the size(in pixels) of the applet display area.

**ALIGN** is an optional attribute that specifies the alignment of the applet.  
The possible value is: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

**VSPACE AND HSPACE** attributes are optional, VSPACE specifies the space, in pixels, about and below the applet. HSPACE VSPACE specifies the space, in pixels, on each side of the applet

**PARAM NAME AND VALUE:** The PARAM tag allows you to specifies applet-specific arguments in an HTML page applets access there attributes with the get Parameter()method.

**Q. Explain <PARAM> tag of applet with suitable example. [S-15]**

To pass parameters to an applet <PARAM... > tag is used. Each <PARAM...> tag has a name attribute and a value attribute. Inside the applet code, the applet can refer to that parameter by name to find its value.

The syntax of <PARAM...> tag is as follows

**<PARAM NAME = name1 VALUE = value1>**

To set up and handle parameters, two things must be done.

1. Include appropriate <PARAM...> tags in the HTML document.
2. Provide code in the applet to parse these parameters.

Parameters are passed on an applet when it is loaded. Generally init() method in the applet is used to get hold of the parameters defined in the <PARAM...> tag.

The getParameter() method, which takes one string argument representing the name of the parameter and returns a string containing the value of that parameter.

**Example**

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

public class hellouser extends Applet
```



```
{
String str;
public void init()
{
str = getParameter("username");
str = "Hello "+ str;
}
public void paint(Graphics g)
{
g.drawString(str,10,100);
}
}
```

```
<HTML>
<Applet code = -hellouser.class|| width = 400 height = 400>
<PARAM NAME = "username" VALUE = abc>
</Applet>
</HTML>
```

**Q. How can parameter be passed to an applet? Write an applet to accept user name in the form of parameter and print „Hello<username>“. [W-15]**

### **Passing Parameters to Applet**

User defined parameters can be supplied to an applet using <PARAM.....> tags. PARAM tag names a parameter the Java applet needs to run, and provides a value for that parameter.

PARAM tag can be used to allow the page designer to specify different colors, fonts, URLs or other data to be used by the applet.

**To set up and handle parameters, two things must be done.**

1. Include appropriate <PARAM..>tags in the HTML document.

The Applet tag in HTML document allows passing the arguments using param tag.

The syntax of <PARAM...> tag

```
<Applet code="AppletDemo" height=300 width=300>
<PARAM NAME = name1 VALUE = value1>
</Applet>
```

**NAME:** attribute name

**VALUE:** value of attribute named by corresponding PARAM NAME.

2. Provide code in the applet to parse these parameters.

The Applet access their attributes using the getParameter method.

The syntax is : **String getParameter(String name);**



---

### Program for an applet to accept user name in the form of parameter and print „Hello<username>“ [W-15]

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

public class hellouser extends Applet
{
String str;
public void init()
{
str = getParameter("username");
str = "Hello "+ str;
}
public void paint(Graphics g)
{
g.drawString(str,10,100);
}
}
```

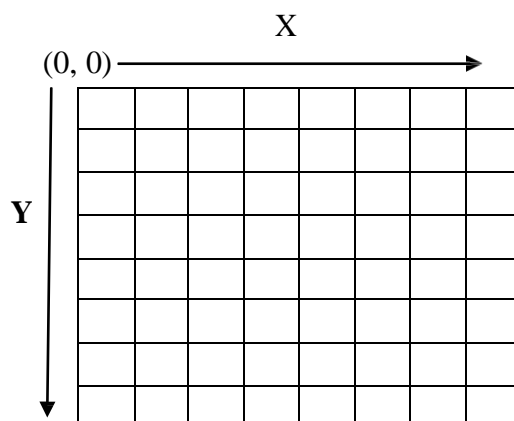
```
<HTML>
<Applet code = -hellouser.class|| width = 400 height = 400>
<PARAM NAME = "username" VALUE = abc>
</Applet>
</HTML>
```

## 5.2. Graphics Programming

Graphics can be drawn with the help of java. java applets are written to draw lines, figures of different shapes, images and text in different styles even with the colours in display.

Every applet has its own area of the screen known as canvas, where it creates the display in the area specified the size of applet's space is decided by the attributes of <APPLET...> tag.

A java applet draws graphical image inside its space using the coordinate system shown in following fig., which shows java's coordinate system has the origin (0, 0) in the upper-left corner, positive x values are to be right, and positive y values are to the bottom. The values of coordinates x and y are in pixels.





---

**Q. Write a simple applet which display message „Welcome to Java“. [W-15]**

**Program:**

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

public class Welcome extends Applet
{
    public void paint( Graphics g)
        {
            g.drawString(-Welcome to javall,25,50);
        }
}

/*<applet code= WelcomeJava width= 300 height=300>
</applet>*/
```

**Step to run an Applet**

1. Write a java applet code and save it with as a class name declared in a program by extension as a .java.  
e.g. from above java code file we can save as a **Welcome.java**
2. Compile the java file in command prompt jdk as shown below  
C:\java\jdk1.7.0\bin> javac Welcome.java
3. After successfully compiling java file, it will create the .class file, e.g Welcome.class. then we have to write applet code to add this class into applet.

4. Applet code

```
<html>
<Applet code= — Welcome.class|| width= 500 height=500>
</applet>
</html>
```

5. Save this file with Welcome.html in ‘\_bin’ library folder.
6. Now write the following steps in command prompt jdk.

```
C:\java\jdk1.7.0\bin> appletviewer Welcome.java
C:\java\jdk1.7.0\bin> appletviewer Welcome.html
(Shows output in applet viewer)
OR
C:\java\jdk1.7.0\bin> Welcome.html
(Shows output in internet browser)
```

**5.2.1. Graphics Class**

The Graphics class of java includes methods for drawing different types of shapes, from simple lines to polygons to text in a variety of fonts.



---

The `paint()` method and a `Graphics` object is used to display text. To draw shapes, drawing methods in `Graphics` class is used which arguments representing end points, corners, or starting locations of a shape as a values in the applet's coordinate system.

<b>Method</b>	<b>Description</b>
<code>clearRect()</code>	Erases a rectangular area of the canvas
<code>copyArea()</code>	Copies a rectangular area of the canvas to another area
<code>drawArc()</code>	Draws a hollow arc.
<code>drawLine()</code>	Draws a straight line
<code>drawOval()</code>	Draws a hollow oval
<code>drawPolygon()</code>	Draws a hollow polygon
<code>drawRect()</code>	Draws a hollow rectangle
<code>drawRoundRect()</code>	Draws a hollow rectangle with rounded corners.
<code>drawstring()</code>	Displays a text string
<code>fillArc()</code>	Draws a filled arc
<code>fillOval()</code>	Draws a filled arc
<code>fillPolygon()</code>	Draws a filled polygon
<code>fillRect()</code>	Draws a filled rectangle
<code>fillRoundRect()</code>	Draws filled rectangle with rounded corners
<code>getColor()</code>	Retrieves the current drawing color
<code>getFont()</code>	Retrieves the currently used font
<code>getFontMetrics()</code>	Retrieves information about the current font.
<code>setColor()</code>	Sets the drawing color
<code>setFont()</code>	Seta fonts.

### 5.2.2. `drawString()` [S-15]

Displaying String:

`drawString()` method is used to display the string in an applet window

**Syntax:**

```
void drawString(String message, int x, int y);
```

where message is the string to be displayed beginning at x, y

**Example:**

```
g.drawString("WELCOME", 10, 10);
```



---

### 5.2.3. Lines and Rectangle.

#### 5.2.3.1. drawLine()

The drawLine () method is used to draw line which takes two pair of coordinates (x1,y1) and (x2, y2) as arguments and draws a line between them. The graphics object g is passed to paint() method.

The syntax is

```
g.drawLine(x1,y1,x2,y2);
```

e.g. g.drawLine(20,20,80,80);

#### 5.2.3.2. drawRect() [W-14, S-15,W-15, S-16 ]

The drawRect() method display an outlined rectangle

**Syntax:** void drawRect(int top, int left, int width, int height)

This method takes four arguments, the first two represents the x and y co- ordinates of the top left corner of the rectangle and the remaining two represent the width and height of rectangle.

**Example:** g.drawRect(10,10,60,50);

**Q. Design an Applet program which displays a rectangle filled with red color and message as “Hello Third year Students” in blue color. [S-16]**

**Program-**

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

public class DrawRectangle extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g.fillRect(10,60,40,30);

        g.setColor(Color.blue);
        g.drawString("Hello Third year Students",70,100);
    }
}
```

```
/* <applet code="DrawRectangle.class" width="350" height="300"> </applet> */
```





---

## 5.2.4. Circle and Ellipse

### 5.2.4.1. drawOval( ) [ W-14, S-15, W-15, S-16]

To draw an Ellipses or circles used drawOval() method can be used.

**Syntax:** void drawOval( int top, int left, int width, int height)

The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by top and left and whose width and height are specified by width and height to draw a circle or filled circle, specify the same width and height the following program draws several ellipses and circle.

**Example:** g.drawOval(10,10,50,50);

### 5.2.4.2. fillOval ( ) [ W-14 ]

Draws an oval within a bounding rectangle whose upper left corner is specified by top, left. Width and height of the oval are specified by width and height.

**Syntax-** void fillOval(int top, int left, int width, int height):

**Example** g.fillOval(10,10,50,50);

**Q. Write a simple applet program which display three concentric circle. [S-16]**

**Program-**

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

public class CircleDemo extends Applet
{
    public void paint (Graphics g)
    {
        g.drawOval(100,100,190,190);
        g.drawOval(115,115,160,160);
        g.drawOval(130,130,130,130);
    }
}

/*<applet code=||CircleDemo.class| height=300 width=200>
</applet>*/
```

(OR)

HTML Source:

```
<html> <applet code=||CircleDemo.class| height=300 width=200>
</applet>
</html>
```

---

**Q. Write a program to design an applet to display three circles filled with three different colors on screen. [ W-14, W-15 ]**

**Program-**

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

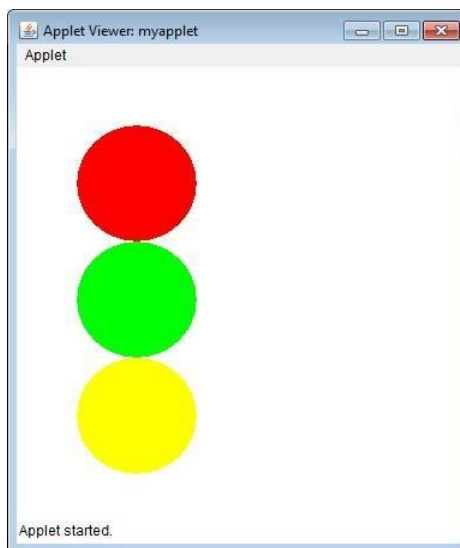
public class myapplet extends Applet
{
    public void paint(Graphics g)
    { g.setColor(Color.red);
      g.fillOval(50,50,100,100);

      g.setColor(Color.green);
      g.fillOval(50,150,100,100);

      g.setColor(Color.yellow);
      g.fillOval(50,250,100,100);
    }
}

/*<applet code=myapplet width= 300 height=300>
</applet>*/
```

**Output**



**5.2.5. Drawing Arcs**

**5.2.5.1. drawArc() [ S-15, W-15 ]**

It is used to draw arc

**Syntax:**

```
void drawArc(int x, int y, int w, int h, int start_angle, int sweep_angle);
```



---

where x, y starting point, w& h are width and height of arc, and start\_angle is starting angle of arc sweep\_angle is degree around the arc

**Example:**

```
g.drawArc(10, 10, 30, 40, 40, 90);
```

## 5.2.6. Drawing polygons

### 5.2.6.1. drawPolygon( ) [W-14, W-15]

drawPolygon() method is used to draw arbitrarily shaped figures.

**Syntax- void drawPolygon(int[ ] xPoints, int[ ] yPoints, int numPoints):**

The polygon's end points are specified by the co-ordinates pairs contained within the x and y arrays. The number of points define by x and y is specified by numPoints.

**Example-**

```
int x[ ] = {10, 170, 80};
int y[ ] = {20, 40, 140};
int n = 3;

g.drawPolygon(x, y, n);
```

**Q. Write the syntax and example for each of following graphics methods:**

1) drawPoly ( ) 2) drawRect ( ) 3) drawOval ( ) 4) fillOval ( )

**For syntax refer above 5.2.3.2 and all.....**

**Example for including all methods in a one program**

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

public class DrawGraphics extends Applet
{
    public void paint(Graphics g)
    {
        int x[ ] = {10, 170, 80};
        int y[ ] = {20, 40, 140};
        int n = 3;
        g.drawPolygon(x, y, n);
        g.drawRect(10, 150, 100, 80);
        g.drawOval(10, 250, 100, 80);
        g.fillOval(10, 350, 100, 80);
    }
}

/*
<applet code = DrawGraphics.class height = 500 width = 400>
</applet>*/
```



### 5.2.7. Setting color of an Applet

Background and foreground color of an applet can be set by using followings methods

```
void setBackground(Color.newColor)
```

```
void setForeground (Color.newColor)
```

where newColor specifies the new color. The class color defines the constant for specific color listed below.

Color.black	Color.white	Color.pink	Color.yellow
Color.lightGray	Color.gray	Color.darkGray	Color.red
Color.green	Color.magenta	Color.orange	Color.cyan

#### Example

```
setBackground(Color.red);
```

```
setForeground (Color.yellow);
```

The following methods are used to retrieve the current background and foreground color.

```
Color getBackground( )
```

```
Color getForeground( )
```

### 5.2.8. Font class

A font determines look of the text when it is painted. Font is used while painting text on a graphics context & is a property of AWT component.

The Font class defines these variables:

Variable	Meaning
String name	Name of the font
float pointSize	Size of the font in points
int size	Size of the font in point
int style	Font style

#### 5.2.8.1. Use of font class [W-14, S-15]

The Font class states fonts, which are used to render text in a visible way. It is used to set or retrieve the screen font.

#### Syntax to create an object of Font class. [W-14]

To select a new font, you must first construct a Font object that describes that font. Font constructor has this general form:



---

### Font(String fontName, int fontStyle, int pointSize)

fontName specifies the name of the desired font. The name can be specified using either the logical or face name.

All Java environments will support the following fonts:

Dialog, DialogInput, Sans Serif, Serif, Monospaced, and Symbol. Dialog is the font used by once system's dialog boxes.

Dialog is also the default if you don't explicitly set a font. You can also use any other fonts supported by particular environment, but be careful—these other fonts may not be universally available.

The style of the font is specified by fontStyle. It may consist of one or more of these three constants:

Font.PLAIN, Font.BOLD, and Font.ITALIC. To combine styles, OR them together.

For example,

Font.BOLD | Font.ITALIC specifies a bold, italics style.

The size, in points, of the font is specified by pointSize.

To use a font that you have created, you must select it using setFont( ), which is defined by Component.

It has this general form:

```
void setFont(Font fontObj)
```

Here, fontObj is the object that contains the desired font

#### 5.2.8.2. Methods of font class

**Q. Describe any three methods of font class with their syntax and example of each. [W-14, S-15 ]**

Sr. No	Methods	Description
1	static Font decode(String <i>str</i> )	Returns a font given its name.
2	boolean equals(Object <i>FontObj</i> ) :	Returns <b>true</b> if the invoking object contains the same font as that specified by <i>FontObj</i> . Otherwise, it returns <b>false</b> .
3	String toString( )	Returns the string equivalent of the invoking font.
4	String getFamily( )	Returns the name of the font family to which the invoking font belongs.
5	static Font getFont(String <i>property</i> )	Returns the font associated with the system property specified by <i>property</i> . <b>null</b> is returned if <i>property</i> does not exist.
6	static Font getFont(String <i>property</i> ,Font <i>defaultFont</i> )	Returns the font associated with the System property specified by <i>property</i> . The font specified by <i>defaultFont</i> is returned if <i>property</i> does not exist.
7	String getFontName( )	Returns the face name of the invoking font.
8	String getName( )	Returns the logical name of the invoking font.



9	int getSize( )	Returns the size, in points, of the invoking font.
10	int getStyle( )	Returns the style values of the invoking font.
11	int hashCode( )	Returns the hash code associated with the invoking object.
12	boolean isBold( )	Returns <b>true</b> if the font includes the <b>BOLD</b> style value. Otherwise, <b>false</b> is returned.
13	boolean isItalic( )	Returns <b>true</b> if the font includes the <b>ITALIC</b> style value. Otherwise, <b>false</b> is returned.
14	boolean isPlain( )	Returns <b>true</b> if the font includes the <b>PLAIN</b> style value. Otherwise, <b>false</b> is returned.

### Example:-

#### //program using equals method

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

public class ss extends Applet
{
    public void paint(Graphics g)
    {
        Font a = new Font ("TimesRoman", Font.PLAIN, 10);
        Font b = new Font ("TimesRoman", Font.PLAIN, 10);

        // displays true since the objects have equivalent settings
        g.drawString(""+a.equals(b),30,60);
    }
}

/*<applet code=|ss.class| height=200 width=200>
</applet>*/
```

#### // program using getFontName,getFamily(),getSize(),getStyle(),.getName()

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

public class font1 extends Applet
{
    Font f, f1;
    String s, msg;
    String fname;
    String ffamily;
    int size;
    int style;
    public void init()
    {
```



```
f= new Font("times new roman",Font.ITALIC,20);
setFont(f);
msg="is interesting";
s="java programming";
fname=f.getFontName();
ffamily=f.getFamily();
size=f.getSize();
style=f.getStyle();
String f1=f.getName();
}
public void paint(Graphics g)
{
g.drawString("font name"+fname,60,44);
g.drawString("font family"+ffamily,60,77);
g.drawString("font size "+size,60,99);
g.drawString("fontstyle "+style,60,150);
g.drawString("fontname "+f1,60,190);
}
}
```

```
/*<applet code=font1.class height=300 width=300>
</applet>*/
```

**Q. Write method to set font of a text and describe its parameters. [S-16]**

The AWT supports multiple type fonts emerged from the domain of traditional type setting to become an important part of computer-generated documents and displays. The AWT provides flexibility by abstracting font-manipulation operations and allowing for dynamic selection of fonts.

Fonts have a family name, a logical font name, and a face name. The family name is the general name of the font, such as Courier. The logical name specifies a category of font, such as Monospaced. The face name specifies a specific font, such as Courier Italic To select a new font, you must first construct a Font object that describes that font.

One Font constructor has this general form:  
Font(String fontName, intfontStyle, intpointSize)

To use a font that you have created, you must select it using setFont( ), which is defined by Component.

It has this general form:  
**void setFont(Font fontObj)**

**Example**

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class SampleFonts extends Applet
{
int next = 0;
Font f;
String msg;
```



```
public void init()  
{  
    f = new Font("Dialog", Font.PLAIN, 12);  
    msg = "Dialog";  
    setFont(f);  
public void paint(Graphics g)  
{  
    g.drawString(msg, 4, 20);  
}  
}
```

**Q. State purpose of get Available Font Family Name ( ) method of graphics environment class.**

**Purpose of getAvailableFontFamilyName() method:**

It returns an array of String containing the names of all font families in this Graphics Environment localized for the specified locale

**Syntax:**

**public abstract String[ ] getAvailableFontFamilyNames(Locale l)**

**Parameters:**

l - a Locale object that represents a particular geographical, political, or cultural region. Specifying null is equivalent to specifying Locale.getDefault().

**Or**

**String[ ] getAvailableFontFamilyNames()**

It will return an array of strings that contains the names of the available font families

### **Important Questions:-**

#### **4 Marks Questions:-**

- 1) Write syntax and example of 1) drawString ( ) 2) drawRect ( ) ; 3) drawOval ( ) 4) drawArc ( ) .
- 2) Describe following states of applet life cycle : a) Initialization state. b) Running state. c) Display state
- 3) State the use of font class. Describe any three methods of font class with their syntax and example of each.
- 4) Differentiate applet and application with any four points.
- 5) State syntax and explain it with parameters for : i) drawRect ( ) ii) drawOval ( )
- 6) Design an Applet program which displays a rectangle filled with red color and message as -Hello Third year Students in blue color.
- 7) Describe applet life cycle with suitable diagram.
- 8) Differentiate between applet and application (any 4 points).
- 9) Write a program to design an applet to display three circles filled with three different colors on screen.
- 10) Explain all attributes available in < applet > tag.



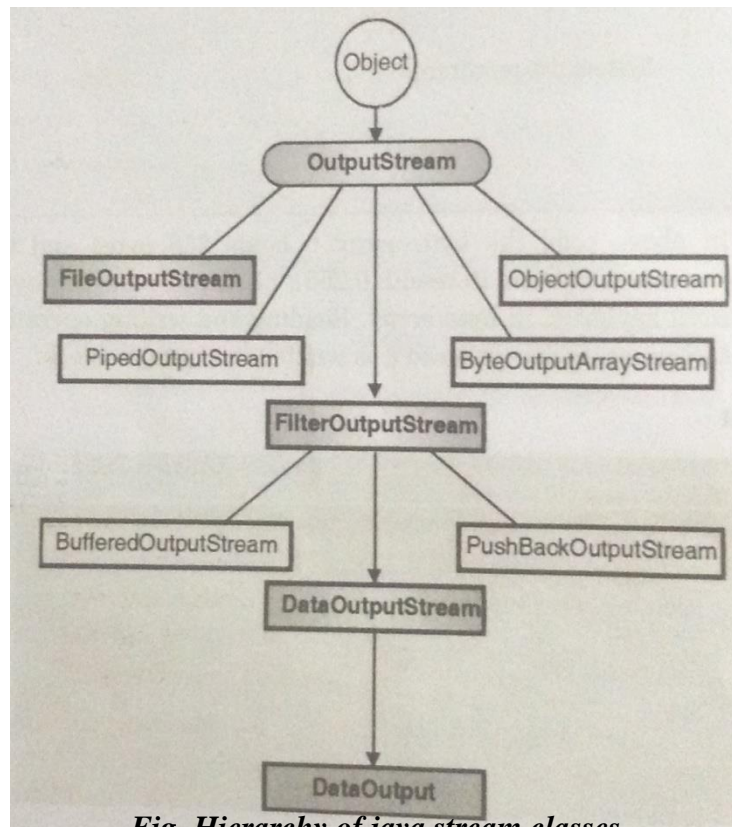


---

### 6 & 8 Marks Questions:-

- 1) Explain <PARAM> Tag of applet with suitable example.
- 2) State the use of font class. Describe any three methods of font class with their syntax and example of each.
- 3) Write a simple applet program which display three concentric circle.
- 4) Write method to set font of a text and describe its parameters.
- 5) Explain <applet> tag with its major attributes only. State purpose of get Available Font Family Name ( ) method of graphics environment class.
- 6) Design an applet which displays three circles one below the other and fill them red, green and yellow color respectively.
- 7) Write the syntax and example for each of following graphics methods : 1) drawPoly( ) 2) drawRect ( ) 3) drawOval ( ) 4) fillOval ( )
- 8) State the use of Font class. Write syntax to create an object of Font class.
- 9) Describe any 3 methods of Font class with their syntax and example of each.
- 10) Write syntax and example of following Graphics class methods : (i) drawOval( ) (ii) drawPolygon( ) (iii) drawArc( ) (iv) drawRect( )
- 11) Differentiate between applet and application and also write a simple applet which display message \_Welcome to Java‘.
- 12) How can parameters be passed to an applet ? Write an applet to accept user name in the form of parameter and print \_Hello < username >‘.

### 6.1. Stream Classes



*Fig. Hierarchy of java stream classes*

1. What are stream classes ? List any two input stream classes from character stream [S-15, S-16]

**Definition:**

The java. IO package contain a large number of stream classes that provide capabilities for processing all types of data. These classes may be categorized into two groups based on the data type on which they operate.

1. Byte stream classes that provide support for handling I/O operations on bytes.
2. Character stream classes that provide support for managing I/O operations on characters.

Character Stream Class can be used to read and write 16-bit Unicode characters. There are two kinds of character stream classes, namely, reader stream classes and writer stream classes



---

### Reader stream classes:-

It is used to read characters from files. These classes are functionally similar to the input stream classes, except input streams use bytes as their fundamental unit of information while reader streams use characters

### Input Stream Classes

1. BufferedReader
2. CharArrayReader
3. InputStreamReader
4. FileReader
5. PushbackReader
6. FilterReader
7. PipeReader
8. StringReader

## 2. What are streams ? Write any two methods of character stream classes. [W-15]

Java programs perform I/O through streams. A stream is an abstraction that either produces or consumes information (i.e it takes the input or gives the output). A stream is linked to a physical device by the Java I/O system.

All streams behave in the same manner, even if the actual physical devices to which they are linked differ. Thus, the same I/O classes and methods can be applied to any type of device.

Java 2 defines two types of streams: byte and character.

Byte streams provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary data.

Character streams provide a convenient means for handling input and output of characters.

They use Unicode and, therefore, can be internationalized. Also, in some cases, character streams are more efficient than byte streams.

### The Character Stream Classes

Character streams are defined by using two class hierarchies. At the top are two abstract classes, **Reader** and **Writer**. These abstract classes handle Unicode character streams. Java has several concrete subclasses of each of these.

### Methods of Reader Class

1) **void mark(int numChars)** : Places a mark at the current point in the input stream that will remain valid until numChars characters are read.

2) **boolean markSupported()** : Returns **true** if **mark()** / **reset()** are supported on this stream.

3) **int read()** :Returns an integer representation of the next available character from the invoking input stream. -1 is returned when the end of the file is encountered.



---

4) **int read(char buffer[ ])** : Attempts to read up to buffer. Length characters into buffer and returns the actual number of characters that were successfully read. -1 is returned when the end of the file is encountered.

5) **abstract int read(char buffer[ ],int offset,int numChars)**: Attempts to read up to numChars characters into buffer starting at buffer[offset], returning the number of characters successfully read.-1 is returned when the end of the file is encountered.

6) **boolean ready( )**: Returns **true** if the next input request will not wait. Otherwise, it returns **false**.

7) **void reset( )**: Resets the input pointer to the previously set mark.

8) **long skip(long numChars)** :- Skips over numChars characters of input, returning the number of characters actually skipped.

9) **abstract void close( )** :- Closes the input source. Further read attempts will generate an **IOException**

### **Writer Class**

**Writer** is an abstract class that defines streaming character output. All of the methods in this class return a **void** value and throw an **IOException** in the case of error

#### **Methods of Writer class are listed below: -**

1) **abstract void close( )** : Closes the output stream. Further write attempts will generate an **IOException**.

2) **abstract void flush( )** : Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.

3) **void write(int ch)**: Writes a single character to the invoking output stream. Note that the parameter is an **int**, which allows you to call **write** with expressions without having to cast them back to **char**.

4) **void write(char buffer[ ])**: Writes a complete array of characters to the invoking output stream

5) **abstract void write(char buffer[ ],int offset, int numChars)** :- Writes a subrange of numChars characters from the array buffer, beginning at buffer[offset] to the invoking output stream.

6) **void write(String str)**: Writes str to the invoking output stream.

7) **void write(String str, int offset,int numChars)**: Writes a sub range of numChars characters from the array str, beginning at the specified offset.

**[\*\*Note: any two methods from above list to be considered]**



---

### 3. What is use of stream classes ? Write any two methods FileReader class.[W-14]

An I/O Stream represents an input source or an output destination.

A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects.

Some streams simply pass on data; others manipulate and transform the data in useful ways. Java's stream based I/O is built upon four abstract classes: InputStream, OutputStream, Reader, Writer.

They are used to create several concrete stream subclasses, the top level classes define the basic functionality common to all stream classes.

InputStream and OutputStream are designed for byte streams and used to work with bytes or other binary objects.

Reader and Writer are designed for character streams and used to work with character or string.

1. `public int read()throws IOException` - Reads a single character.
2. `public int read(char[ ] cbuf, int offset, int length) throws IOException` - Reads characters into a portion of an array.
3. `public void close()throws IOException` - Closes the stream and releases any system resources associated with it. Once the stream has been closed, further `read()`, `ready()`, `mark()`, `reset()`, or `skip()` invocations will throw an `IOException`. Closing a previously closed stream has no effect
4. `public boolean ready()throws IOException` - Tells whether this stream is ready to be read. An `InputStreamReader` is ready if its input buffer is not empty, or if bytes are available to be read from the underlying byte stream
5. `public void mark(int readAheadLimit) throws IOException` -Marks the present position in the stream. Subsequent calls to `reset()` will attempt to reposition the stream to this point. Not all character-input streams support the `mark()` operation.
6. `public void reset()throws IOException` - Resets the stream. If the stream has been marked, then attempt to reposition it at the mark. If the stream has not been marked, then attempt to reset it in some way appropriate to the particular stream, for example by repositioning it to its starting point. Not all character-input streams support the `reset()` operation, and some support `reset()` without supporting `mark()`.

### 4. Write any two methods of File and FileInputStream class each.[S-15, W-15]

#### File Class Methods

1. `String getName()` - returns the name of the file.
2. `String getParent()` - returns the name of the parent directory.



**3. boolean exists( )** - returns true if the file exists, false if it does not.

**4. void deleteOnExit( )** -Removes the file associated with the invoking object when the Java Virtual Machine terminates.

**5. boolean isHidden( )**-Returns true if the invoking file is hidden. Returns false otherwise.

### **FileInputStream Class Methods:**

**1. int available( )** - Returns the number of bytes of input currently available for reading.

**2. void close( )** - Closes the input source. Further read attempts will generate an IOException.

**3. void mark(int numBytes)** -Places a mark at the current point in the inputstream that will remain valid until numBytes bytes are read.

**4. boolean markSupported( )** -Returns true if mark( )/reset( ) are supported by the invoking stream.

**5. int read( )** - Returns an integer representation of the next available byte of input. -1 is returned when the end of the file is encountered.

**6. int read(byte buffer[ ])** - Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. -1 is returned when the end of the file is encountered.

### **5. Explain serialization in relation with stream class. [W-14,W-15, S-16]**

Serialization is the process of writing the state of an object to a byte stream. This is useful when you want to save the state of your program to a persistent storage area, such as a file. At a later time, you may restore these objects by using the process of deserialization.

Serialization is also needed to implement Remote Method Invocation (RMI). RMI allows a Java object on one machine to invoke a method of a Java object on a different machine. An object may be supplied as an argument to that remote method. The sending machine serializes the object and transmits it. The receiving machine deserializes it.

#### **Example:**

Assume that an object to be serialized has references to other objects, which, in turn, have references to still more objects. This set of objects and the relationships among them form a directed graph. There may also be circular references within this object graph. That is, object X may contain a reference to object Y, and object Y may contain a reference back to object X. Objects may also contain references to themselves. The object serialization and deserialization facilities have been designed to work correctly in these scenarios. If you attempt to serialize an object at the top of an object graph, all of the other referenced objects are recursively located and serialized. Similarly, during the process of deserialization, all of these objects and their references are correctly restored.



---

**6. Write a program to copy contents of one file to another file using character stream class.[S-15]**

```
import java.io.*;
class CopyData
{
    public static void main(String args[ ])
    {
        //Declare input and output file stream

        FileInputStream fis= null; //input stream

        FileOutputStream fos=null; //output Stream
        //Declare a variable to hold a byte

        byte byteRead;
        try
        {
            // connect fis to in.dat
            fis=new FileInputStream(-in.dat);
            // connect fos to out.dat
            fos= new FileOutputStream(-out.dat);
            //reading bytes from in.dat and write to out.dat

            do
            {
                byteRead =(byte)fis.read( );
                fos.write(byteRead);
            }
            while(byteRead != -1);
        }

        Catch(FileNotFoundException e)

        {

            System.out.println(-file not found);

        }

        Catch(IOException e)

        {

            System.out.pritln(e.getMessage( ));

        }

        finally // close file

        {
```



```
try
{
fis.close( );
fos.close( );
}
Catch(IOException e)
{ }
}
}
```

**7. What is use of ArrayList Class ? State any three methods with their use from ArrayList.[W-15, S-16]**

**Use of ArrayList class:**

1. ArrayList supports dynamic arrays that can grow as needed.
2. ArrayList is a variable-length array of object references. That is, an ArrayList can dynamically increase or decrease in size. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.

**Methods of ArrayList class :**

1. **void add(int index, Object element)** Inserts the specified element at the specified position index in this list. Throws `IndexOutOfBoundsException` if the specified index is out of range (`index < 0 || index > size()`).
2. **boolean add(Object o)** Appends the specified element to the end of this list.
3. **boolean addAll(Collection c)** Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws `NullPointerException` if the specified collection is null.
4. **boolean addAll(int index, Collection c)** Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws `NullPointerException` if the specified collection is null.
5. **void clear()** Removes all of the elements from this list.
6. **Object clone()** Returns a shallow copy of this ArrayList.





---

**7. boolean contains(Object o)** Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element *e* such that  $(o == null ? e == null : o.equals(e))$ .

**8. void ensureCapacity(int minCapacity)** Increases the capacity of this `ArrayList` instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

**9. Object get(int index)** Returns the element at the specified position in this list. Throws `IndexOutOfBoundsException` if the specified index is out of range ( $index < 0$  ||  $index \geq size()$ ).

**10. int indexOf(Object o)** Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

**11. int lastIndexOf(Object o)** Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

**12. Object remove(int index)** Removes the element at the specified position in this list. Throws `IndexOutOfBoundsException` if index out of range ( $index < 0$  ||  $index \geq size()$ ).

**13. protected void removeRange(int fromIndex, int toIndex)** Removes from this List all of the elements whose index is between `fromIndex`, inclusive and `toIndex`, exclusive.

**14. Object set(int index, Object element)** Replaces the element at the specified position in this list with the specified element. Throws `IndexOutOfBoundsException` if the specified index is out of range ( $index < 0$  ||  $index \geq size()$ ).

**15. int size()** Returns the number of elements in this list.

**16. Object[] toArray()** Returns an array containing all of the elements in this list in the correct order. Throws `NullPointerException` if the specified array is null.

**17. Object[] toArray(Object[] a)** Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

**18. void trimToSize()** Trims the capacity of this `ArrayList` instance to be the list's current size.



---

**8. Write syntax and function of following methods of Date class :**

**i) getTime () ii) getDate () [ S-15]**

The **Date** class encapsulates the current date and time.

**i. getTime():**

Syntax:

**long getTime()**

Returns the number of milliseconds that have elapsed since January 1, 1970.

**ii. getDate()**

Syntax:

**public int getDate()**

Returns the day of the month. This method assigns days with the values of 1 to 31.

**9. Write syntax and function of following methods of date class :**

**1) setTime () 2) getDay () [W-14]**

**1. setTime():**

void setTime(long time):

the parameter time - the number of milliseconds.

Sets this Date object to represent a point in time that is time milliseconds after January 1, 1970 00:00:00 GMT

**2. getDay()**

int getDay():

Returns the day of the week represented by this date.

The returned value (0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday) represents the day of the week that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

**10. Write any four mathematical functions used in Java.[W-14]**

**1) min() :**

Syntax: static int min(int a, int b)

Use: This method returns the smaller of two int values.

**2) max() :**

Syntax: static int max(int a, int b)

Use: This method returns the greater of two int values.

**3) sqrt()**

Syntax: static double sqrt(double a)

Use : This method returns the correctly rounded positive square root of a double



value.

**4) pow() :**

Syntax: static double pow(double a, double b)

Use : This method returns the value of the first argument raised to the power of the second argument.

**5) exp()**

Syntax: static double exp(double a)

Use : This method returns Euler's number e raised to the power of a double value.

**6) round() :**

Syntax: static int round(float a)

Use : This method returns the closest int to the argument.

**7) abs()**

Syntax: static int abs(int a)

Use : This method returns the absolute value of an int value.

**11. What is use of setclass ? Write a program using setclass.[W-14]**

The Set interface defines a set. It extends Collection and declares the behavior of a collection that does not allow duplicate elements. Therefore, the add( ) method returns false if an attempt is made to add duplicate elements to a set.

The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

The methods declared by Set are summarized in the following table

Sr.No	Methods	Description
1	add()	Adds an object to the collection
2	clear()	Removes all objects from the collection
3	contains()	Returns true if a specified object is an element within the collection
4	isEmpty()	Returns true if the collection has no elements
5	iterator()	Returns an Iterator object for the collection which may be used to retrieve an object
6	remove()	Removes a specified object from the collection
7	size()	Returns the number of elements in the collection



**Following is the example to explain Set functionality:**

```
import java.util.*;
public class SetDemo
    {
    public static void main(String args[])
    {
    int count[] = {34, 22,10,60,30,22};
    Set<Integer> set = new HashSet<Integer>();
    try{
    for(int i = 0; i<5; i++){
    set.add(count[i]);
    }
    System.out.println(set);
    TreeSet sortedSet = new TreeSet<Integer>(set);
    System.out.println("The sorted list is:");
    System.out.println(sortedSet);
    System.out.println("The First element of the set is: "+ (Integer)sortedSet.first());

    System.out.println("The last element of the set is: "+ (Integer)sortedSet.last());
    }
    catch(Exception e){ }
    }
    }
```

**Executing the program.**

```
[34, 22, 10, 30, 60]
The sorted list is:
[10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60
```

## **12. State syntax and describe any two methods of map class.[S-16]**

The Map Classes Several classes provide implementations of the map interfaces. A map is an object that stores associations between keys and values, or key/value pairs. Given a key, you can find its value. Both keys and values are objects. The keys must be unique, but the values may be duplicated. Some maps can accept a null key and null values, others cannot.

### **Methods:**

**void clear** // removes all of the mapping from map

**booleancontainsKey(Object key)** //Returns true if this map contains a mapping for the specified key.

**Boolean conainsValue(Object value)**// Returns true if this map maps one or more keys to the specified value

**Boolean equals(Object o)** //Compares the specified object with this map for equality