
Experiment No: 01

Aim: Programs on Operators, Arithmetic Promotion, Method Calling.

Objectives: 1. To write simple Java applications.
2. To use input and output statements.

Theory:

Operators:

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. Java supports a rich set of operators.

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and Decrement Operators
6. Conditional Operators
7. Bitwise Operators
8. Special Operators

1. Arithmetic Operators:- Different arithmetic operators supported by java language are listed below:-

Symbol Used	Operation Performed
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

---

2. Relational Operators:- These operators are used to compare the values of the operands that must

be compitable, in order to facilitate decision-making.

Symbol Used	Operation Performed
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not Equal to

3. Logical Operators:- Java has the following three logical operators.

Symbol Used	Operation Performed
!	Logical NOT
&&	Logical AND
||	Logical OR

4. Assignment Operator :- Assignment operators are used to assign the result of an expression to a

variable.

Assignment Operator:- '='

Ex:-

c = a+b.

---

5. Increment / Decrement Operator:- Java supports two very useful operators increment (++) and

decrement operator (--), these are not generally found in other languages. The increment operator adds

1 from the operand, while decrement operator subtracts 1 from the operand. Both are unary ope rators

and takes the following form:-

++m (pre incrementation) or m++ (post incrementation)

--m (pre decrementation) or m-- (post decrementation)

6. Conditional Operator (?:) :- A ternary operator pair "?:" is available in the java to construct

conditional expression of the form Exp1 ? Exp2 : Exp3. This the only operator that take 3 operands.

Result = Testing Expression : Exp1 : Exp2

If Testing Expression is True perform expression 1 and assign to result. If Testing Expression is

False perform expression 2 and assign to resut.

7. Bitwise Operator:-These operator are used for testing the bits or for shifting them right or left.

Bitwise operators may not be appiled to float or double.

Symbol Used	Operation Performed
>>	Shift-right
<<	Shift-left
>>>	Shift-right with zero fill
-	One's complement
&	Bitwise AND

!	Bitwise OR
^	Bitwise Exclusive OR

Main() Method:

It is the starting point of every Java application. The parentheses after the identifier main indicate that it's a program building block called a method. Java class declarations normally contain one or more methods. For a Java application, one of the methods must be called main and must be defined as shown ; otherwise, the Java Virtual Machine (JVM) will not execute the application. Methods perform tasks and can return information when they complete their tasks. Keyword void indicates that this method will not return any information. It has to have this exact signature in order to be able to run our program.

public static void main(String[] args) The arguments we get inside the method are the arguments that we will get when running the program with parameters. It's an array of strings.

Source Code:

i) Program on operators

```
class Main {

  public static void main(String[] args) {


    // declare variables

    int a = 12, b = 5;


    // addition operator

    System.out.println("a + b = " + (a + b));


    // subtraction operator

    System.out.println("a - b = " + (a - b));
```

```
    // multiplication operator
    System.out.println("a * b = " + (a * b));

    // division operator
    System.out.println("a / b = " + (a / b));

    // modulo operator
    System.out.println("a % b = " + (a % b));
  }
}
```

Output:

a + b = 17

a - b = 7

a * b = 60

a / b = 2

a % b = 2

ii) Program on Method calling

```
import java.util.*;
public class StaticMethodCallExample
{
public static void main(String args[])
{
int a;
//calling static method of the Math class
```

```
a=Math.min(23,98);
System.out.println("Minimum number is: " + a);
}
}
```

Output:

Minimum number is: 23

Experiment No: 02

Aim : Programs on Classes: String and Math.

Objectives : To understand how to use the Math methods and string methods with literals and variables.

Theory:

Java provides a number of classes and methods for use by programmers. A few of those items are described below:

Math Class:

The Math class methods are static methods that are called from the class name (e.g. Math.sqrt()). Each method takes at least one parameter and performs the required operations on it.

- Math.sqrt(num) - returns the square root of the num
- Math.pow(base, exp) - returns base raised to the exp power
- Math.min(num1, num2) - returns the smallest of the two numbers
- Math.max(num1, num2) - returns the largest of the two numbers
- Math.round(num) - returns the rounded value of num (.5+ rounds up, under .5 rounds down)
- Math.ceil(num) - returns the next highest integer - rounds up (4.2 returns 5)
- Math.floor(num) - returns the next lowest integer - rounds down (6.8 returns 6)
- Math.abs(num) - returns the absolute value of num (-5 returns 5)
- Math.random() - returns a random value between 0 and 1.0
- Math.PI - a constant used to represent 3.1415...

String Class:

The String class methods must be called from a string variable. In other words, String str="programming", must first be declared and the methods would be called from the str variable (str.toUpperCase()). Many string method operate on indicies, a numbered order of characters. The important thing to note is that strings start counting at 0. So, in str, the index of the character p is 0.

- length - a property (not method) that returns the number of characters in the string (str.length returns 11)
- indexOf(string) - returns the starting index of the first occurence of string in str, -1 if not found (str.indexOf("gram") returns 3)

- lastIndexOf(string) - does the same as indexOf, but counts backwards from the end
- charAt(num) - returns the character at the specified index (str.charAt(4) returns "r")
- toUpperCase() - returns the string as all upper case (str.toUpperCase() returns PROGRAMMING)
- toLowerCase() - returns the string as all lower case
- substring(start, [end]) - returns a substring of the string starting at start, up to but not including end. If end is not specified, the remainder of the string is included (str.substring(3, 7) returns "gram", str.substring(7) returns "ming")

Source Code:

i) Program on Math Class

```
public class JavaMathExample1
{
   public static void main(String[] args)
   {
      double x = 28;
      double y = 4;

      // return the maximum of two numbers
      System.out.println("Maximum number of x and y is: " +Math.max(x, y));

      // return the square root of y
      System.out.println("Square root of y is: " + Math.sqrt(y));

      //returns 28 power of 4 i.e. 28*28*28*28
      System.out.println("Power of x and y is: " + Math.pow(x, y));

      // return the logarithm of given value
```

```
        System.out.println("Logarithm of x is: " + Math.log(x));
        System.out.println("Logarithm of y is: " + Math.log(y));

        // return the logarithm of given value when base is 10
        System.out.println("log10 of x is: " + Math.log10(x));
        System.out.println("log10 of y is: " + Math.log10(y));

        // return the log of x + 1
        System.out.println("log1p of x is: " +Math.log1p(x));

        // return a power of 2
        System.out.println("exp of a is: " +Math.exp(x));

        // return (a power of 2)-1
        System.out.println("expm1 of a is: " +Math.expm1(x));
    }
}
```

Output:

```
Maximum number of x and y is: 28.0
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
```

```
log1p of x is: 3.367295829986474
exp of a is: 1.446257064291475E12
expm1 of a is: 1.446257064290475E12
```

ii) Program on String class

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

    // create first string
    String first = "Java ";
    System.out.println("First String: " + first);

    // create second
    String second = "Programming";
    System.out.println("Second String: " + second);

    // join two strings
    String joinedString = first.concat(second);
    System.out.println("Joined String: " + joinedString);
  }
}
```

Output:

```
First String: Java
Second String: Programming
Joined String: Java Programming
```

Experiment No: 03

Title : **Write a program to demonstrate following Function concepts**
i) Function overloading
ii) Constructors

Objectives : i) To understand the concept of function calling and overloading.
ii) To understand functioning of constructor.

Theory:

Function Overloading:

Function Overloading in Java occurs when there are functions having the same name but have different numbers of parameters passed to it, which can be different in data like int, double, float and used to return different values are computed inside the respective overloaded method. Function overloading is used to reduce complexity and increase the efficiency of the program by involving more functions that are segregated and can be used to distinguish among each other with respect to their individual functionality. Overloaded functions are related to compile-time or static polymorphism. There is also a concept of type conversion, which is basically used in overloaded functions used to calculate the conversion of type in variables.

Different Ways of Method Overloading in Java

- Changing the Number of Parameters.
- Changing Data Types of the Arguments.
- Changing the Order of the Parameters of Methods

Constructors:

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.
In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory. It is a special type of method which is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called.

Now let us come up with the syntax for the constructor being invoked at the time of object or instance creation.

```
class A
{
  .......

  // A Constructor
  A() {
  }

  .......
}

// We can create an object of the above class
// using the below statement. This statement
// calls above constructor.
A obj = new A();
```

Types of Constructors in Java

- No-argument constructor
- Parameterized Constructor
- Default Constructor

Source Code:

i)Program on Function Overloading

```
public class Sum {


    // Overloaded sum(). This sum takes two int parameters
```

```
public int sum(int x, int y)
{
    return (x + y);
}

// Overloaded sum(). This sum takes three int parameters
public int sum(int x, int y, int z)
{
    return (x + y + z);
}

// Overloaded sum(). This sum takes two double parameters
public double sum(double x, double y)
{
    return (x + y);
}

// Driver code
public static void main(String args[])
{
    Sum s = new Sum();
    System.out.println(s.sum(10, 20));
```

```
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}
```

Output:

30

60

31.0

ii) Program on constructor

Source Code:

```
class Student{

    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n)
   {
    id = i;
    name = n;
    }

    //method to display the values
    void display()
   {
          System.out.println(id+" "+name);
   }
```

```
    public static void main(String args[]){
    //creating objects and passing values
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Output:

111 Karan
222 Aryan

Experiment No: 04

Aim: Programs on Control Statements.

i) Java program to find whether the given no. is positive or negative.

ii) Java Program to print weekdays according to integers from 1 to 7.

Objectives: To review and apply the concepts involved in the usage of control statements to store and modify data according to the given condition.

Theory:

Control Structure:

Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
   - if statements
   - if – else statement
   - if-else ladder statement
   - nested if statement
   - switch statement
2. Loop statements
   - do while loop
   - while loop
   - for loop
   - for-each loop
3. Jump statements
   - break statement
   - continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

- Simple if statement
- if-else statement
- if-else-if ladder
- Nested if-statement
- Switch Statement

1. Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Booleanexpression and enables the program to enter a block of code if the expression evaluates to true.

```
if(condition) {

    statement 1; //executes when condition is true
   }
```

2. if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax:

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false

}
```

4. Nested if-statement

In nested if-statements, the if statement can contain a if or if-else statement inside another if or else-if statement.

Syntax:

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
}
```

5. Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Syntax:

```
switch (expression){

case value1: statement1; break;.
.
.
case valueN: statementN; break; default:
default statement;
}
```

Java for loop:
In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

Syntax:

```
for(initialization, condition, increment/decrement) {
//block of statements
}
```

Java for-each loop:
Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

Syntax:

```
for(data_type var : array_name/collection_name){
//statements
}
```

Java while loop:
The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.
The syntax of the while loop is given below.

```
while(condition){
```

//looping statements
}

Java do-while loop:
The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do
{
//statements
} while (condition);
```

Java break statement:
As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

Java continue statement:
Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Source Code:
i) Java program to find whether the given no. is positive or negative.

```
public class CheckPositiveOrNegativeExample1
{
public static void main(String[] args)
{
//number to be check
int num=912;
//checks the number is greater than 0 or not
if(num>0)
```

```
{
System.out.println("The number is positive.");
}
//checks the number is less than 0 or not
else if(num<0)
{
System.out.println("The number is negative.");
}
//executes when the above two conditions return false
else
{
System.out.println("The number is zero.");
}
}
}
```

Output:

The number is positive.

ii) Java Program to print weekdays according to integers from 1 to 7.

```
import java.util.Scanner;
public class Main {


  public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Input number: ");
      int day = in.nextInt();

      System.out.println(getDayName(day));
   }

   // Get the name for the Week
   public static String getDayName(int day) {
      String dayName = "";
      switch (day) {
```

```
            case 1: dayName = "Monday"; break;
            case 2: dayName = "Tuesday"; break;
            case 3: dayName = "Wednesday"; break;
            case 4: dayName = "Thursday"; break;
            case 5: dayName = "Friday"; break;
            case 6: dayName = "Saturday"; break;
            case 7: dayName = "Sunday"; break;
            default:dayName = "Invalid day range";
        }

        return dayName;
    }
}
```

Output:

```
Input number: 3
Wednesday
```

Experiment No: 05

Aim: Programs on dealing with Arrays.

i) Java Program to Find Largest Element in an Array.

ii) Java Program to Print Even and Odd Elements in an Array.

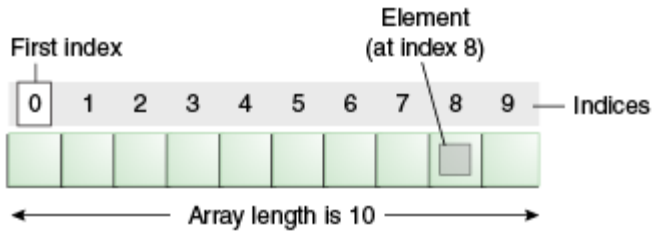
Objectives: To review and apply the concepts involved in the usage of Arrays to store and modify data.

Theory:

Array:

An array is a data structure to store a collection of values of the same type. The data can be accessed using its integer index. Since all the data is stored in contiguous memory locations, an array is always initialized with its size and it cannot be changed while the program is running.

The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.



This is an array of 10 elements. All the elements are integers and homogeneous. The green box below the array is called the index, which always starts from zero and goes up to n-1 elements. In this case, as there are 10 elements, the index is from zero to nine

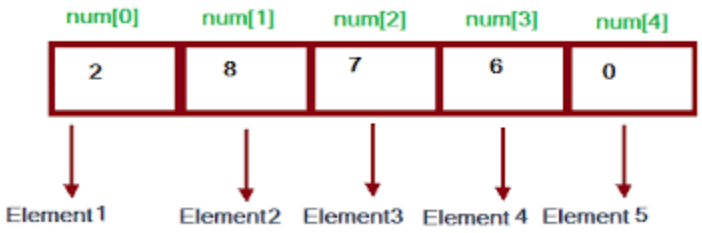
Types of Array in java:

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

---

1. **Single Dimensional Array:**



Declaration of Array:

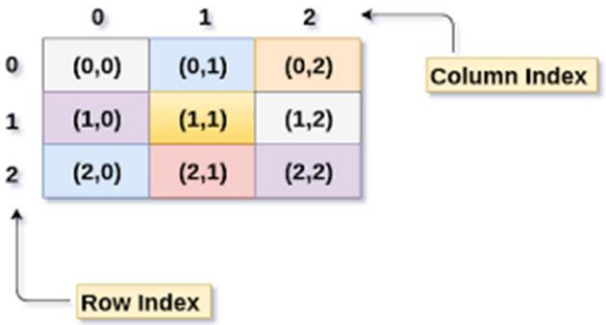
dataType[] arrayRefVar; // preferred way. Or
dataType arrayRefVar[]; // works but not preferred way.

dataType[] arrayRefVar = new dataType[arraySize];

dataType[] arrayRefVar = {value0, value1, ..., valuek};

2. **Two-dimensional Array:**

Two-dimensional arrays store the data in rows and columns:



In this, the array has 3 rows and 3 columns. The index starts from 0,0 in the left-upper corner to 2,2 in the right lower corner.

Declaration:

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

```
dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

int[][] arr=new int[3][3];

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;
```

3. Three Dimentional Array declaration in Java:

```
data-type[ ][ ][ ] variableName;

variableName = new data-type[size1][size2][size3];

data-type[ ][ ][ ] variableName = new data-type[size1][size2][size3];
```

Source Code:

i) Java Program to Find Largest Element in an Array

```
class Max {
   public static void main(String[] args) {
      int a[]={1,2,3,4,5};
      int max;
```

```
        for(int i=0;i<a.length;i++)
        {

        System.out.println(a[i]);
           }

        max = a[0];
        for(int i = 0; i < a.length; i++)
        {
           if(max < a[i])
           {
              max = a[i];
           }
        }
        System.out.println("Maximum value:"+max);
     }
}
```

Output:

```
1
2
3
4
5
Maximum value:5
```

---

ii) Java Program to Print Even and Odd Elements in an Array

```
class EvenOdd {
   public static void main(String[] args) {
      int a[]={1,2,3,4,5,6,7,8,9,10};
      int max;
      System.out.println("Even Nos");

      for(int i=0;i<a.length;i++)
      {
        if(a[i]%2==0)
      System.out.println(a[i]);
         }
          System.out.println("Odd Nos");

      for(int i=0;i<a.length;i++)
      {
        if(a[i]%2!=0)
      System.out.println(a[i]);
         }
 }
}
```

Output:

```
Even Nos
2
4
6
8
10
Odd Nos
1
3
5
7
9
```

Experiment No: 06

Aim: Write a Java program that illustrates the following

i) Creation of simple package.

ii) Accessing a package.

Objectives: To understand the mechanism to encapsulate a group of classes, sub packages and interfaces and how to access them.

Theory:

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code.

Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

Built-in Packages:

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.The library contains components for managing input, database programming, and much more. The library is divided into packages and classes. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

To use a class or a package from the library, you need to use the **import keyword:**

Syntax :

import package.name.Class; // Import a single class

import package.name.*; // Import the whole package

Import a Class If you find a class you want to use, for example, the Scanner class, which is used to get user input, write the following code: Example import java.util.Scanner; In the example above, java.util is a package, while Scanner is a class of the java.util package.

User-defined Packages:

To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer:

Example

└── root

└── mypack

└── MyPackageClass.java

To create a package, use the package keyword:

MyPackageClass.java

```
package mypack;

class MyPackageClass {

public static void main(String[] args) {

 System.out.println("This is my package!");

 }

}
```

Source Code:

i) Program on creation of simple package

```
package MyPackage;

public class Compare {
  int num1, num2;

  public Compare(int n, int m) {
     num1 = n;
     num2 = m;
}
public void getmax(){
```

```
    if ( num1 > num2 ) {
       System.out.println("Maximum value of two numbers is " + num1);
     }
      else {
       System.out.println("Maximum value of two numbers is " + num2);
      }
}

public static void main(String args[]) {
       Compare current[] = new Compare[3];

       current[1] = new Compare(5, 10);
       current[2] = new Compare(123, 120);

         for(int i=1; i < 3 ; i++)
            {
            current[i].getmax();
            }
      }
}
```

Output:

Maximum value of two numbers is 10
Maximum value of two numbers is 123

ii) Program on accesing a package

```
package Ncer;
import MyPackage.Compare;

public class Demo{
   public static void main(String args[]) {
       int n=10, m=10;
       Compare current = new Compare(n, m);
       if(n != m) {
           current.getmax();
       }
```

```
        else {
            System.out.println("Both the values are same");
        }
    }
    }
```

Output:

1

Both the values are same

Experiment No: 07

Aim: Programs on Inheritance and Polymorphism.

Objectives: To understand and apply the concepts of polymorphism and inheritance according to problem specification given.

Theory:

Inheritance:

Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class. In Java, inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Objectives of Inheritance in Java:

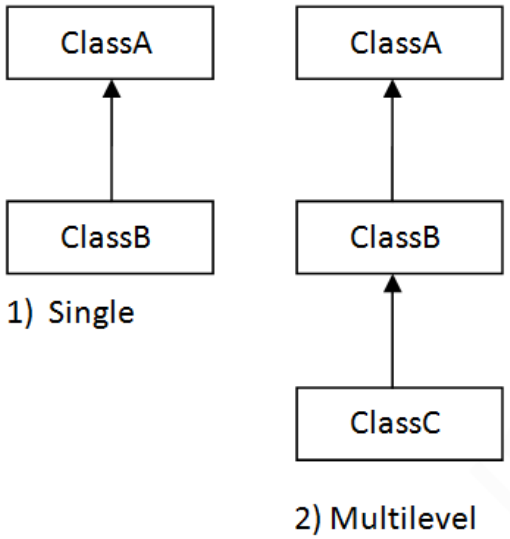
- **Code Reusability:** The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class's code.
- **Method Overriding:** Method Overriding is achievable only through Inheritance. It is one of the ways by which java achieves Run Time Polymorphism.
- **Abstraction:** The concept of abstract where we do not have to provide all details is achieved through inheritance. Abstraction only shows the functionality to the user.

The syntax of Java Inheritance:

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

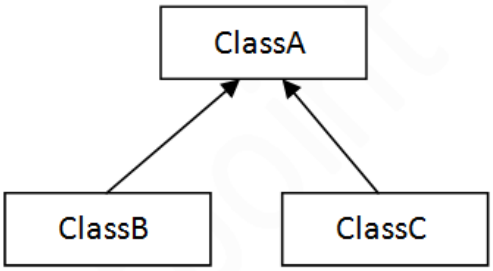
Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

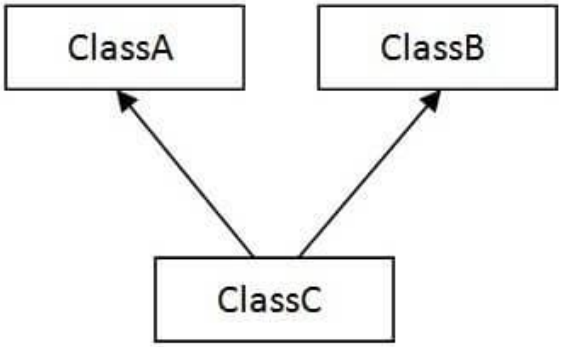


1) Single

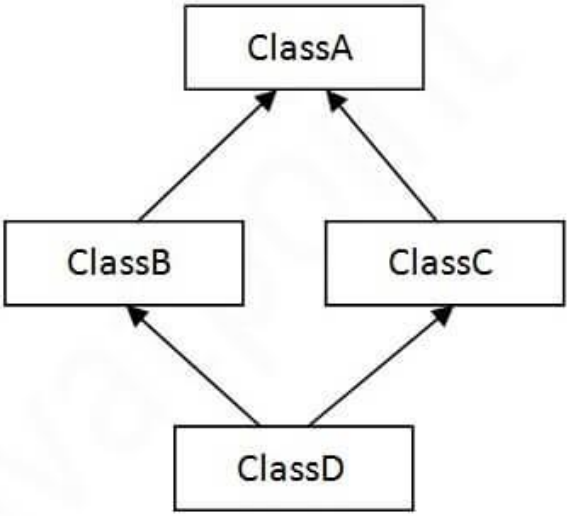
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

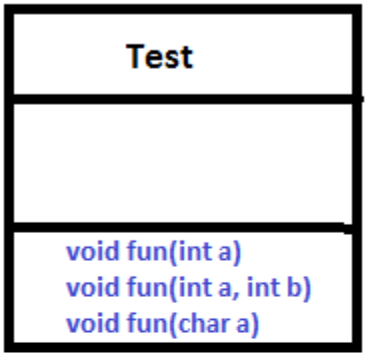
Polymorphism:

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word "poly" means many and "morphs" means forms, So it means many forms.

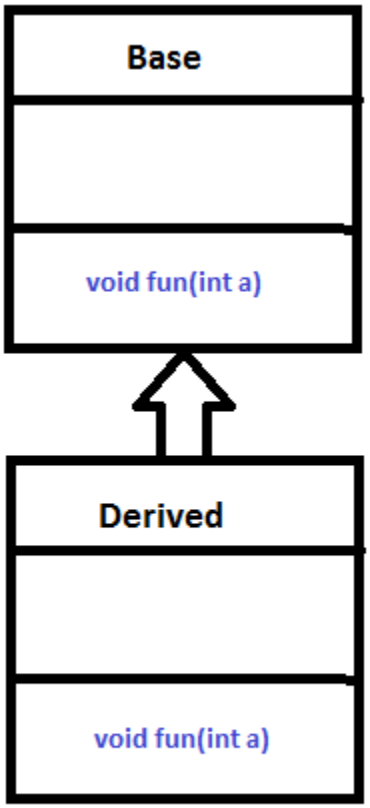
Types of polymorphism

In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism



Overloading



Overriding

1) **Compile-time Polymorphism(Method Overloading)**: When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by change in the number of arguments or/and a change in the type of arguments.

---

2) **Run Time Polymorphism:** It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding. **Method overriding**, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.

Source Code:

i) Program on Inheritance

```
class Animal{
void eat(){
System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){
System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){
System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
```

```
d.eat();

}}
```

Output:

```
weeping...
barking...
eating...
```

ii) Program on Polymorphism

```
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
class TestPolymorphism2{
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

Output:

drawing rectangle...
drawing circle...
drawing triangle...

Experiment No: 08

Aim: Programs on Exception Handling.

Objectives: To understand the mechanism of how to handle the exceptions occurred in runtime so that program will execute properly**.**

Theory:

Exceptions:

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Exceptions can be categorized in two ways:

1. Built-in Exceptions
   - Checked Exception
   - Unchecked Exception
2. User-Defined Exceptions

Let us discuss the above-defined listed exception that is as follows:

Built-in Exceptions:

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

- **Checked Exceptions:** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

- **Unchecked Exceptions:** The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

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

Source Code:

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


    try {


      // code that generate exception
      int divideByZero = 5 / 0;
      System.out.println("Rest of code in try block");
    }


    catch (ArithmeticException e) {
      System.out.println("ArithmeticException => " + e.getMessage());
    }
  }
}
```

Output

ArithmeticException => / by zero

Experiment No: 09

Aim: Program on Java script client side scripting.

Objectives: To be able to design more interactive and faster web application using client side scripting language.

Theory:

Client-side scripting, like JavaScript, can be embedded into the page on the client's browser. This script will allow the client's browser to alleviate some of the burden on your web server when running a web application. Client-side scripting is source code that is executed on the client's browser instead of the web-server, and allows for the creation of faster and more responsive web applications.

Client-side scripting (embedded scripts) is code that exists inside the client's HTML page. This code will be processed on the client machine and the HTML page will NOT perform a PostBack to the web-server. Traditionally, client-side scripting is used for page navigation, data validation and formatting. The language used in this scripting is JavaScript. JavaScript is compatible and is able to run on any internet browser.

The two main benefits of client-side scripting are:

1. The user's actions will result in an immediate response because they don't require a trip to the server.
2. Fewer resources are used and needed on the web-server.

Javascript:

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

JavaScript is used to create interactive websites. It is mainly used for:

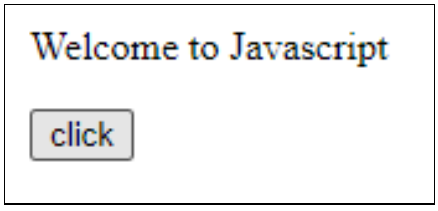
- Client-side validation,
- Dynamic drop-down menus,

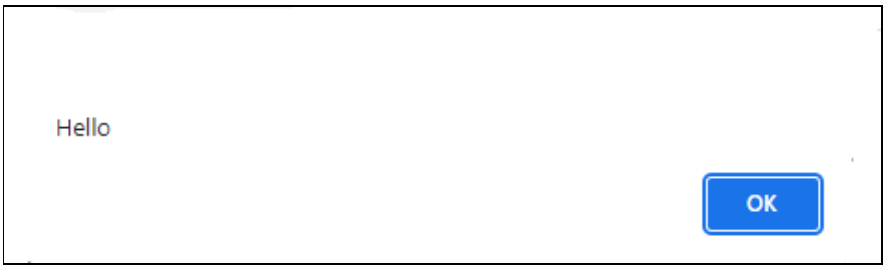
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

Source Code:

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello");
}
</script>
</head>
<body>
<p>Welcome to Javascript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

Output:




Hello
OK

Experiment No: 10

Aim: Programs on Java script Operators, Comparisons, Statements, Loops, Events.

Objectives: To be able to design more interactive and faster web application using JavaScript.

Theory:

JavaScript operators are symbols that are used to perform operations on operands

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

JavaScript Arithmetic Operators:

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators:

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators:

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Statements:

The programming instructions written in a program in a programming language are known as statements.

Order of execution of Statements is the same as they are written.

- Semicolons:
- Code Blocks
- White Space
- Line Length and Line Breaks:
- Keywords

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When JavaScript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

---

Source Code:

iii) Program on Operators

```
<html>
  <body>

    <script type = "text/javascript">
        var a = 33;
        var b = 10;
        var c = "Test";
        var linebreak = "<br />";

        document.write("a + b = ");
        result = a + b;
        document.write(result);
        document.write(linebreak);

        document.write("a - b = ");
        result = a - b;
        document.write(result);
        document.write(linebreak);

        document.write("a / b = ");
        result = a / b;
        document.write(result);
        document.write(linebreak);

        document.write("a % b = ");
        result = a % b;
        document.write(result);
        document.write(linebreak);

    </script>

    Set the variables to different values and then try...
  </body>
</html>
```

Output:

a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
Set the variables to different values and then try...

iv) Program on Comparison Operators

```
<html>
  <body>
    <script type = "text/javascript">
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

        document.write("(a == b) => ");
        result = (a == b);
        document.write(result);
        document.write(linebreak);

        document.write("(a < b) => ");
        result = (a < b);
        document.write(result);
        document.write(linebreak);

        document.write("(a > b) => ");
        result = (a > b);
        document.write(result);
        document.write(linebreak);

        document.write("(a != b) => ");
        result = (a != b);
        document.write(result);
        document.write(linebreak);

        document.write("(a >= b) => ");
        result = (a >= b);
        document.write(result);
        document.write(linebreak);
```

```
        document.write("(a <= b) => ");
        result = (a <= b);
        document.write(result);
        document.write(linebreak);
    </script>

  </body>
</html>
```

Output:

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true
```

v) Program on Events

```
<!DOCTYPE html>

<html>

<body>

<h2>JavaScript HTML Events</h2>

<p>Click "Try it" to execute the displayDate() function.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>
```

```
<script>

document.getElementById("myBtn").onclick = displayDate;

function displayDate() {

  document.getElementById("demo").innerHTML = Date();

}

</script>

</body>

</html>
```

Output:

Click "Try it" to execute the displayDate() function.

Try it

Fri Nov 04 2022 11:42:32 GMT+0530 (India Standard Time)

vi) Program on loops

```
<html>
  <body>
    <script type = "text/javascript">
      <!--
        var count;
        document.write("Starting Loop" + "<br />");

        for(count = 0; count < 10; count++) {
```

```
            document.write("Current Count : " + count );
            document.write("<br />");
          }
          document.write("Loop stopped!");
        //-->
      </script>

    </body>
</html>
```

Output:

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```