**JAVA**

Java is a class-based object-oriented programming language.

* OOPS Concept in Java

1) Class

The class is one of the Basic concepts of OOPs which is a group of similar entities. It is only a logical component and not the physical entity.

2) Object

An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object is one of the Java OOPs concepts which contains both the data and the function.

3) Inheritance

Inheritance is one of the Basic Concepts of OOPs in which one object acquires the properties and behaviors of the parent object. It’s creating a parent-child relationship between two classes.

4) Polymorphism

Polymorphism refers to one of the OOPs concepts in Java which is the ability of a variable, object or function to take on multiple forms.

5) Abstraction

Abstraction is one of the OOP Concepts in Java which is an act of representing essential features without including background details

6) Encapsulation

Encapsulation is one of the best Java OOPs concepts of wrapping the data under a single code. In this OOPs concept, the variables of a class are always hidden from other classes.

* Types of Variables

1)Local Variables - are a variable that are declared inside the body of a method

2)Instance Variables - are defined without the STATIC keyword.

3)Static Variables - are initialized only once, at the start of the program execution. These variables should be initialized first, before the initialization of any instance variables

**IF**

If is a conditional statement. Used to execute a set of commands by checking upon a certain condition.

Eg:- If(a>b){}

**IF-ELSE**

Statements inside else is executed if “IF” condition turns out to be false.

Eg:- if(a>1){}

Else{}

**CREATING AN ARRAY**

int[] numbers = {3, 9, 5, -5};

Scanner statement to take input from user

// create an object of Scanner class

Scanner input = new Scanner(System.in);

System.out.println("Enter a number");

int number = input.nextInt();

**WHILE**

While(condition)

{

}

If condition is true statements inside while gets executed.

**DO WHILE**

Do{

}

While(condition);

Statements inside do gets executed initially and then while condition is checked for further iteration.

Continue statement

The Java continue statement **stops one iteration in a loop and continues to the next iteration**. This statement lets you skip particular iterations without stopping a loop entirely

Eg:-

First:

{]

Continue first;

**ARRAYS**

// Short form

// (can only be used as initializer in a declaration)

int[] arr1 = { 1, 2, 77 };

// Initialize with 10 zeroes

int[] arr2 = new int[10];

// General form

int[] arr3 = new int[] { 1, 2, 77 };

**PRE AND POST INCREMENT**

++i returns the value after it is incremented, while i++ return the value before it is incremented

**CONSTRUCTORS**

Java 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.

If a constructor is declared as private, then **its objects are only accessible from within the declared class**. You cannot access its objects from outside the constructor class.

Java compiler automatically creates a default constructor (Constructor with no arguments) in case no constructor is **present in the java class**

Eg:

Main{ java}

Main(string langage){cprogram}

// call constructor with no parameter

Main obj1 = new Main();

// call constructor with a single parameter

Main obj2 = new Main("Python");

Obj1.getname(); --- gives output java

Obj2.getname();----gives output python

**STATIC**

When a variable **is declared as static, then a single copy of the variable is created and shared among all objects at a class level**. Static variables are, essentially, global variables. All instances of the class share the same static variable.

**THIS**

this can be used to:

* Invoke current class constructor
* Invoke current class method
* Return the current class object
* Pass an argument in the method call
* Pass an argument in the constructor call

class this\_Test

{

 int val1; int val2;

// Parameterized constructor

    this\_Test(int val1, int val2)

    {

        this.val1 = val1 + val1;

        this.val2 = val2 + val2;

    }

   void display()

    {

          System.out.println("Value val1 = " + val1 + " Value val2 = " + val2);

    }

}

class Main{

       public static void main(String[] args)

       {

            this\_Test object = new this\_Test(5,10);

            object.display();

       }

}

**SUPER**

The super keyword in Java is **a reference variable which is used to refer immediate parent class object**.

class Parentclass

{

//Overridden method

void display(){

System.out.println("Parent class method");

}

}

class Subclass extends Parentclass

{

//Overriding method

void display(){

System.out.println("Child class method");

}

void printMsg(){

//This would call Overriding method

display();

//This would call Overridden method

super.display();

}

public static void main(String args[]){

Subclass obj= new Subclass();

obj.printMsg();

}

**OVERLOADING**

Java method overloading- With method overloading, multiple methods can have the same name with different parameters

Example:

static int MethodInt(int x, int y) {

return x + y;

}

static double MethodDouble(double x, double y) {

return x + y;

}

public static void main(String[] args) {

int myNum1 = MethodInt(8, 5);

double myNum2 = MethodDouble(4.3, 6.26);

System.out.println("int: " + myNum1);

System.out.println("double: " + myNum2);}

**METHOD OVERRIDING**

* Java method overriding- Method overriding is one of the way by which java achieve Run Time Polymorphism.

Example:

// Base Class

class Parent {

void show() {

System.out.println("Parent's show()");

}

}

//inherited class

class Child extends Parent {

// This method overrides show() of Parent

//override

Void show()

{

System.out.println("Child's show()");

}

}

//Driver class

Class main{

public static void main(String[] args)

{

// If a Parent type reference refer

// to a Parent object, then Parent's

//show() is called

Parent obj1 = new Parent();

obj1.show(); // If a Parent type reference refers

// to a Child object Child's show()

// is called. This is called RUN TIME POLYMORHISM

Parent obj2 = new Child();

Parent obj2 = new Child();

obj2.show();

}

}

**Java 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.

Eg:-

Public class main(){

…

..

Public main(){}

Public static void mian(){}

Here constructor gets invoked when we create an object of the same class.

**Java static keyword**

A static method can be accessed without creating an object of the class first.

**Static methods cannot be overridden** because they are not dispatched on the object instance at runtime.

The main() method is static **so that JVM can invoke it without instantiating the class**.

**Java super keyword**

The super keyword refers to superclass objects.

The super keyword in Java is **a reference variable that is used to refer parent class objects**. The super() in Java is a reference variable that is used to refer parent class constructors. super can be used to call parent class' variables and methods. super() can be used to call parent class' constructors only.

**Java this keyword**

The this keyword refers to the current object in a method or constructor.

The this keyword refers to **the current object in a method or constructor**. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

**Access Modifiers**

The public keyword is an access modifier, meaning that it is used to set the access level for classes, attributes, methods and constructors.

Public: The code is accessible for all classes

Private: The code is only accessible within the declared class.

Protected: The code is accessible in the same package and subclasses. You will learn more about subclasses and superclasses in the Inheritance chapter.