***OOPs in JAVA***

**What is Object-Oriented Programming?**

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

Object-oriented programming (OOP) is a fundamental programming paradigm based on the concept of “objects”. These objects can contain data in the form of fields (often known as attributes or properties) and code in the form of procedures (often known as methods).

The core concept of the object-oriented approach is to break complex problems into smaller objects.

## What is a Class in Java?

A class is a structure that defines the data and the methods to work on that data. When you write programs in the Java language, all program data is wrapped in a class, whether it is a class you write or a class you use from the Java platform API libraries.

Classes in the Java platform API libraries define a set of objects that share a common structure and behavior.

A class is defined as a collection of objects. You can also think of a class as a blueprint from which you can create an individual object.

To create a class, we use the keyword class.

Syntax:

class ClassName {

// fields

// methods

}

## What is an Object in Java?

Object is termed as an instance of a class, and it has its own state, behavior and identity.

An object is an entity in the real world that can be distinctly identified.

An object consists of:

* **A unique identity:** Each object has a unique identity, even if the state is identical to that of another object.
* **State/Properties/Attributes:** State tells us how the object looks or what properties it has.
* **Behavior:**Behavior tells us what the object does.

### **Examples of object states and behaviors in Java:**

#### **Example 1:**

* Object: car.
* State: color, brand, weight, model.
* Behavior: break, accelerate, turn, change gears.

#### **Example 2:**

* Object: house.
* State: address, color, location.
* Behavior: open door, close door, open blinds.

### **Syntax of an object:**

public class Number {

int y = 10;

public static void main(String[] args) {

Number myObj = new Number();

System.out.println(myObj.y);

}

}

## How Access Modifiers Work in Java?

In Java, access modifiers are keywords that set the accessibility of classes, methods, and other members.

These keywords determine whether a field or method in a class can be used or invoked by another method in another class or sub-class.

Access modifiers may also be used to restrict access.

In Java, we have four types of access modifiers, which are:

* Default
* Public
* Private
* Protected

### **Default Access Modifier**

The default access modifier is also called package-private. You use it to make all members within the same package visible, but they can be accessed only within the same package.

Note that when no access modifier is specified or declared for a class, method, or data member, it automatically takes the default access modifier.

**Example:**

class SampleClass

{

void output()

{

System.out.println("Hello World! This is an Introduction to OOP.");

}

}

class Main

{

public static void main(String args[])

{

SampleClass obj = new SampleClass();

obj.output();

}

}

**Explanation:**

* **void output():** When there is no access modifier, the program automatically takes the default modifier.
* **SampleClass obj = new SampleClass**();:  This line of code allows the program to access the class with the default access modifier.
* **obj.output();:** This line of code allows the program to access the class method with the default access modifier.
* **Output**: Hello World! This is an Introduction to OOP.

### **Public Access Modifier**

The public access modifier allows a class, a method, or a data field to be accessible from any class or package in a Java program. The public access modifier is accessible within the package as well as outside the package.

In general, a public access modifier does not restrict the entity at all.

**Example:**

// Car.java file

// public class

public class Car {

// public variable

public int tireCount;

// public method

public void display() {

System.out.println("I am a Car.");

System.out.println("I have " + tireCount + " tires.");

}

}

// Main.java

public class Main {

public static void main( String[] args ) {

// accessing the public class

Car car = new Car();

// accessing the public variable

car.tireCount = 4;

// accessing the public method

car.display();

}

}

**Explanation:**

* The public class **Car** is accessed from the Main class.
* The public variable **tireCount** is accessed from the Main class.
* The public method **display()** is accessed from the Main class.

### **Private Access Modifier**

The **private access modifier**is an access modifier that has the lowest accessibility level. This means that the methods and fields declared as private are not accessible outside the class. They are accessible only within the class which has these private entities as its members.

You may also note that the private entities are not visible even to the subclasses of the class.

**Example:**

class SampleClass

{

private String activity;

}

public class Main

{

public static void main(String[] main)

{

SampleClass task = new SampleClass();

task.activity = "We are learning the core concepts of OOP.";

}

}

**Explanation:**

* **private String activity**: The private access modifier makes the variable “activity” a private one.
* **SampleClass task = new SampleClass();**: We have created an object of SampleClass.
* **task.activity = "We are learning the core concepts of OOP.";**: On this line of code we are trying to access the private variable and field from another class (which can never be accessible because of the private access modifier).
* When we run the above program, we will get an error message. This is because we are trying to access the private variable and field from another class.
* So, the best way to access these private variables is to use the getter and setter methods.
* Getters and setters are used to protect your data, particularly when creating classes. When we create a getter method for each instance variable, the method returns its value while a setter method sets its value.

**Example:**

class SampleClass

{

private String task;

// This is the getter method.

public String getTask()

{

return this.task;

}

// This is the setter method.

public void setTask(String task)

{

this.task= task;

}

}

public class Main

{

public static void main(String[] main)

{

SampleClass task = new SampleClass();

// We want to access the private variable using the getter and setter.

task.setTask("We are learning the core concepts of OOP.");

System.out.println(task.getTask());

}

}

**Explanation:**

As we have a private variable named **task** in the above example, we have used the methods **getTask()** and **setTask()** in order to access the variable from the outer class. These methods are called getter and setter in Java.

We have used the setter method (**setTask()**) to assign value to the variable and the getter method (**getTask()**) to access the variable.

### **Protected Access Modifier**

When methods and data members are declared protected, we can access them within the same package as well as from subclasses.

We can also say that the protected access modifier is somehow similar to the default access modifier. It is just that it has one exception, which is its visibility in subclasses.

**Note:** Classes cannot be declared protected. This access modifier is generally used in a parent-child relationship.

**Example:**

// Multiplication.java

package learners;

public class Multiplication

{

protected int multiplyTwoNumbers(int a, int b)

{

return a\*b;

}

}

// Test.java

package javalearners;

import learners.\*;

class Test extends Multiplication

{

public static void main(String args[])

{

Test obj = new Test();

System.out.println(obj.multiplyTwoNumbers(2, 4));

}

} //output: 8

**Explanation:**

In this example, the class **Test** which is present in another package is able to call the  multiplyTwoNumbers() method, which is declared protected.

The method is able to do so because the Test class extends class Addition and the protected modifier allows the access of protected members in subclasses (in any packages).

## What are Constructors in Java?

A constructor in Java is similar to a method that is invoked when an object of the class is created.

Unlike Java methods, a constructor has the same name as that of the class and does not have any return type.

**Syntax:**

class Test {

Test() {

// constructor body

}

}

**Example:**

class Main {

private String name;

// constructor

Main() {

System.out.println("Constructor Called:");

name = "PK”;

}

public static void main(String[] args) {

// constructor is invoked while

// creating an object of the Main class

Main obj = new Main();

System.out.println(“The name is “ + obj.name);

}

}

### **Types of Constructors:**

In Java, constructors can be divided into three types:

* No-Arg Constructor
* Parameterized Constructor
* Default Constructor

1. Java No-Arg Constructors

Similar to methods, a Java constructor may or may not have any parameters (arguments).

If a constructor does not accept any parameters, it is known as a no-argument constructor. For example,

private Constructor() {

// body of the constructor

}

Example: Java Private No-arg Constructor

class Main {

int i;

// constructor with no parameter

private Main() {

i = 5;

System.out.println("Constructor is called");

}

public static void main(String[] args) {

// calling the constructor without any parameter

Main obj = new Main();

System.out.println("Value of i: " + obj.i);

}

}

**this Keyword in Java**

In Java, this keyword is used to refer to the current object inside a method or a constructor. It means this is nothing but the reference to the current object.

class Main {

int instVar;

Main(int instVar){

this.instVar = instVar;

System.out.println("this reference = " + this);

}

public static void main(String[] args) {

Main obj = new Main(8);

System.out.println("object reference = " + obj);

}

}

## Use of this Keyword

There are various situations where this keyword is commonly used.