Java OOP(Object Oriented Programming) Concepts

Last Updated : 04 Jan, 2025

Object-Oriented Programming or **Java OOPs** concept refers to programming languages that use objects in programming. They use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks you assign.

Object-oriented programming aims to implement real-world entities like **inheritance**, **hiding**, **polymorphism**, etc. in programming. The main aim of OOPs is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Example:

```
// Use of Object and Classes in Java
import java.io.*;

class Numbers {
    // Properties
    private int a;
    private int b;

    // Methods
    public void sum() { System.out.println(a + b); }

    public void sub() { System.out.println(a - b); }

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

```
16
17          // Creating Instance of Class
18          // Object
19          Numbers obj = new Numbers();
20
21          // Assigning Values to the Properties
22          obj.a = 1;
23          obj.b = 2;
24
25          // Using the Methods
26          obj.sum();
27          obj.sub();
28      }
29  }
```

Output

```
3
-1
```

It is a simple example showing a class Numbers containing two variables which can be accessed and updated only by instance of the object created.

Java Class

A Class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. Using classes, you can create multiple objects with the same behavior instead of writing their code multiple times. This includes classes for objects occurring more than once in your code. In general, class declarations can include these components in order:

1. **Modifiers**: A class can be public or have default access (Refer to this for details).
2. **Class name:** The class name should begin with the initial letter capitalized by convention.
3. **Body:** The class body is surrounded by braces, { }.

Java Object

An **Object** is a basic unit of Object-Oriented Programming that represents real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. The objects are what perform your code, they are the part of your code visible to the viewer/user. An object mainly consists of:

1. **State**: It is represented by the attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by the methods of an object. It also reflects the response of an object to other objects.
3. **Identity**: It is a unique name given to an object that enables it to interact with other objects.
4. **Method:** A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping it, which is why they are considered **time savers**. In Java, every method must be part of some class, which is different from languages like C, C++, and Python.

Example:

```
// Java Program to demonstrate
// Use of Class and Objects

// Class Declared
public class GFG {

      // Properties Declared
    static String Employee_name;
    static float Employee_salary;

      // Methods Declared
    static void set(String n, float p) {
        Employee_name  = n;
        Employee_salary  = p;
    }

    static void get() {
        System.out.println("Employee name is: "
+Employee_name );
```

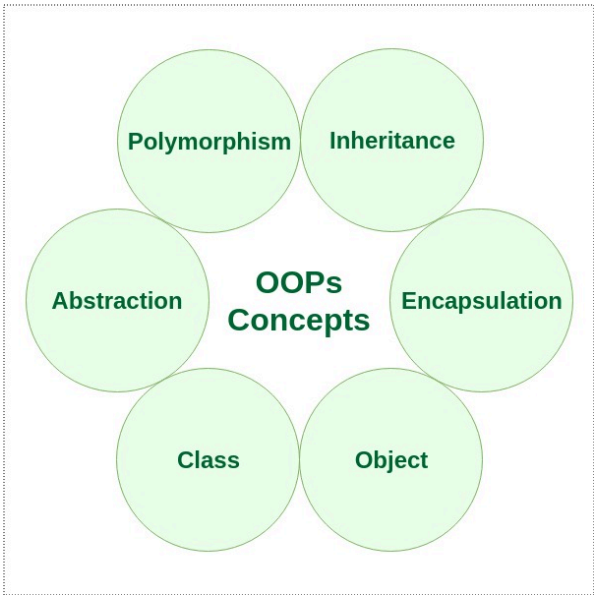
```
        System.out.println("Employee CTC is: " +
Employee_salary);
    }

      // Main Method
    public static void main(String args[]) {
        GFG.set("Rathod Avinash", 10000.0f);
        GFG.get();
    }
}
```

Output

```
Employee name is: Rathod Avinash
Employee CTC is: 10000.0
```

For more information, please refer to the article – **Classes and Object**.



Method and Method Passing

A method is a collection of statements that perform specific tasks and return a result to the caller. It can be declared with or without arguments, depending on the requirements. A method can take input values, perform operations, and return a result.

```
// Class Method and Method Passing
import java.io.*;

class Student {
```

```
5
6       // Properties Declared
7       int id;
8       String name;
9
10      // Printing Student
11      public void printStudent()
12      {
13          System.out.println("Id:" + id);
14          System.out.println("Name:" + name);
15      }
16  }
17
18  class GFG {
19      public static void main(String[] args)
20      {
21          Student obj = new Student();
22
23          obj.id = 1;
24          obj.name = "ABC";
25
26          obj.printStudent();
27      }
28  }
```

Output

```
Id:1
Name:ABC
```

4 Pillars of Java OOPs Concepts

1. Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or non-essential units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
Data Abstraction may also be defined as the process of identifying only the required characteristics of an object, ignoring the irrelevant details. The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the object.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the car speed or applying brakes will stop the car, but he does not know how on pressing the accelerator, the speed is actually increasing. He does not know about the inner mechanism of the car or the implementation of the accelerators, brakes etc. in the car. This is what abstraction is.

Note: In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Example:

```
// abstract class
abstract class GFG {
    // abstract methods declaration
```

```
    abstract void add();
    abstract void mul();
    abstract void div();
}
```

To learn more about the Abstraction refer to the Abstraction in Java article

2. Encapsulation

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

- Technically, in encapsulation, the variables or the data in a class is hidden from any other class and can be accessed only through any member function of the class in which they are declared.
- In encapsulation, the data in a class is hidden from other classes, which is similar to what **data-hiding** does. So, the terms “encapsulation” and “data-hiding” are used interchangeably.
- Encapsulation can be achieved by declaring all the variables in a class as private and writing public methods in the class to set and get the values of the variables.

Example:

```
// Encapsulation using private modifier

// Employee class contains private data
// called employee id and employee name
class Employee {

    private int empid;
    private String ename;

    // Setter methods
```

```
    public void set_id(int empid) {
      this.empid = empid;
    }

    public void set_name(String ename)
    {
        this.ename = ename;
    }

    // Getter methods
    public int get_id() {
      return empid;
    }

    public String get_name() {
      return ename;
    }
}

public class Geeks {

    public static void main(String args[])
    {
        Employee e = new Employee();
        e.set_id(78);
        e.set_name("John");

        System.out.println("Employee id: " +
e.get_id());
        System.out.println("Employee Name: "
                           + e.get_name());
    }
}
```

Output

```
Employee id: 78
Employee Name: John
```

To learn more about topic refer to **Encapsulation in Java** article.

3. 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. We are achieving inheritance by using **extends** keyword. Inheritance is also known as "**is-a**" relationship.

Let us discuss some frequently used important terminologies:

- **Superclass:** The class whose features are inherited is known as superclass (also known as base or parent class).
- **Subclass:** The class that inherits the other class is known as subclass (also known as derived or extended or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example:

```
//base class or parent class or super class
class A{
  //parent class methods
  void method1(){}
  void method2(){}
}

//derived class or child class or base class
class B extends A{  //Inherits parent class methods
  //child class methods
  void method3(){}
  void method4(){}
}
```

To learn more about topic refer to Inheritance in Java article.

4. Polymorphism

It refers to the ability of object-oriented programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities. The ability to appear in many forms is called **polymorphism**.

Example:

```
sleep(1000) //millis
sleep(1000,2000) //millis,nanos
```

Types of Polymorphism

Polymorphism in Java is mainly of 2 types as mentioned below:

1. Method Overloading
2. Method Overriding

Method Overloading and Method Overriding

1. Method Overloading: Also, known as compile-time polymorphism, is the concept of Polymorphism where more than one method share the same name with different signature(Parameters) in a class. The return type of these methods can or cannot be same.

2. Method Overriding: Also, known as run-time polymorphism, is the concept of Polymorphism where method in the child class has the same name, return-type and parameters as in parent class. The child class provides the implementation in the method already written.

Below is the implementation of both the concepts:

```
// Java Program to Demonstrate
// Method Overloading and Overriding

// Parent Class
class Parent {
```

```
6
7       // Method Declared
8       public void func(){
9           System.out.println("Parent Method func");
10      }
11
12      // Method Overloading
13      public void func(int a){
14          System.out.println("Parent Method func " +
    a);
15      }
16  }
17
18  // Child Class
19  class Child extends Parent {
20
21      // Method Overriding
22      @Override
23        public void func(int a){
24          System.out.println("Child Method " + a);
25      }
26  }
27
28  // Main Method
29  public class Main {
30      public static void main(String args[]){
31          Parent obj1 = new Parent();
32          obj1.func();
33          obj1.func(5);
34
35          Child obj2 = new Child();
36          obj2.func(4);
37      }
38  }
```

Output

```
Parent Method func
Parent Method func 5
Child Method 4
```

To know more about the topic refer the **Polymorphism in Java** article.

Advantage of OOPs over Procedure-Oriented Programming Language

Object-oriented programming (OOP) offers several key advantages over procedural programming:

- **OOP promotes code reusability**: By using objects and classes, you can create reusable components, leading to less duplication and more efficient development.
- **OOP enhances code organization**: It provides a clear and logical structure, making the code easier to understand, maintain, and debug.
- **OOP supports the DRY (Don't Repeat Yourself) principle**: This principle encourages minimizing code repetition, leading to cleaner, more maintainable code. Common functionalities are placed in a single location and reused, reducing redundancy.
- **OOP enables faster development**: By reusing existing code and creating modular components, OOP allows for quicker and more efficient application development

Conclusion

The Object Oriented Programming (OOPs) concept in Java is a powerful way to organize and write code. It uses key ideas like classes, objects, inheritance, polymorphism, encapsulation, and abstraction to create flexible and reusable code.

By using the Java OOPs concept, programmers can build complex applications more efficiently, making the code easier to manage, understand, and modify. Overall, Java's OOPs concepts help in creating robust and scalable software solutions.

FAQs – Java OOPs Concepts

What is OOPs concept in Java?

> *OOPs (Object-Oriented Programming) is a programming paradigm based on the concept of objects, which can contain data in the form of fields (attributes or properties) and code in the form of*

procedures (methods or functions). In Java, OOPs concepts include encapsulation, inheritance, polymorphism, and abstraction.

Why is OOPs important in Java?

OOPs helps in organizing and structuring code in a more manageable way, making it easier to maintain and scale Java applications. It also promotes code reusability, modularity, and flexibility, leading to efficient and robust software development.

What are the main principles of OOPs in Java?

The main principles of OOPs in Java are encapsulation, inheritance, polymorphism, and abstraction. Encapsulation ensures that the internal state of an object is hidden and can only be accessed through public methods. Inheritance allows one class to inherit properties and behavior from another. Polymorphism enables objects to be treated as instances of their parent class. Abstraction focuses on hiding the implementation details and showing only the necessary information to the outside world.

How is OOPs implemented in Java?

In Java, OOPs is implemented through classes and objects. A class serves as a blueprint for creating objects, which are instances of that class. Each object has its own set of attributes (variables) and methods (functions). By following OOPs concepts like encapsulation, inheritance, polymorphism, and abstraction, Java developers can design well-structured and maintainable code.

What are the advantages of using OOPs in Java?

Some advantages of using OOPs in Java include code reusability, modularity, flexibility, scalability, and easier maintenance. OOPs enables developers to model real-world entities as objects, leading to more intuitive and organized code. It also supports

features like inheritance and polymorphism, which enhance the extensibility and readability of Java applications.

Can you provide an example of OOPs concept implementation in Java?

Sure! An example of OOPs concept implementation in Java is creating a 'Car' class with attributes like 'make', 'model', and 'year', along with methods like 'start()', 'accelerate()', and 'stop()'. By instantiating objects from the 'Car' class and calling its methods, we can simulate the behavior of different car instances in a structured and object-oriented manner.

Next Article

Classes and Objects in Java

Similar Reads

Best Practices of Object Oriented Programming (OOP)

As the name suggests, Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programmi...

5 min read

Brief Overview & Comparison of Object-Oriented Programming fro...

In this article, you will get the ability to think how really OOP works in Java through C. Through C, you will understand the concept of...

3 min read

7 OOP Design Principles Java Programmers Should Learn in 2023

Java is one of the most widely-used object-oriented programming languages today, known for its robustness, cross-platform compatibility,...

15+ min read

Spring Boot - Difference Between AOP and OOP

AOP(Aspect-Oriented Programming) complements OOP by enabling modularity of cross-cutting concerns. The Key unit of Modularity(breakin...

3 min read

Why Java is not a purely Object-Oriented Language?

Pure Object Oriented Language or Complete Object Oriented Language are Fully Object Oriented Language that supports or have features that...

3 min read

Why C++ is partially Object Oriented Language?

The basic thing which are the essential feature of an object oriented programming are Inheritance, Polymorphism and Encapsulation. Any...

3 min read

Object-Oriented Design (OOD) - System Design

A crucial method for system design is object-oriented design (OOD), which places an intense focus on scalability, modularity, and reusability....

12 min read

Spring Boot - AOP(Aspect Oriented Programming)

The Java applications are developed in multiple layers, to increase security, separate business logic, persistence logic, etc. A typical Java...

4 min read

Collator compare(Object, Object) method in Java with Example

The compare() method of java.text.Collator class is used to compare the strength of two objects and it will return 0, positive and negative value a...