



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:

```
1  // Use of Object and Classes in Java
2  import java.io.*;
3
4  class Numbers {
5      // Properties
6      private int a;
7      private int b;
8
9      // Methods
10     public void sum() { System.out.println(a + b); }
11
12     public void sub() { System.out.println(a - b); }
13
14     public static void main(String[] args)
15     {
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:

```
1  // Java Program to demonstrate
2  // Use of Class and Objects
3
4  // Class Declared
5  public class GFG {
6
7      // Properties Declared
8      static String Employee_name;
9      static float Employee_salary;
10
11     // Methods Declared
12     static void set(String n, float p) {
13         Employee_name = n;
14         Employee_salary = p;
15     }
16
17     static void get() {
18         System.out.println("Employee name is: " +Employee_name);
19         System.out.println("Employee CTC is: " +
20         Employee_salary);
21     }
22
23     // Main Method
24     public static void main(String args[]) {
25         GFG.set("Rathod Avinash", 10000.0f);
26         GFG.get();
27     }
```

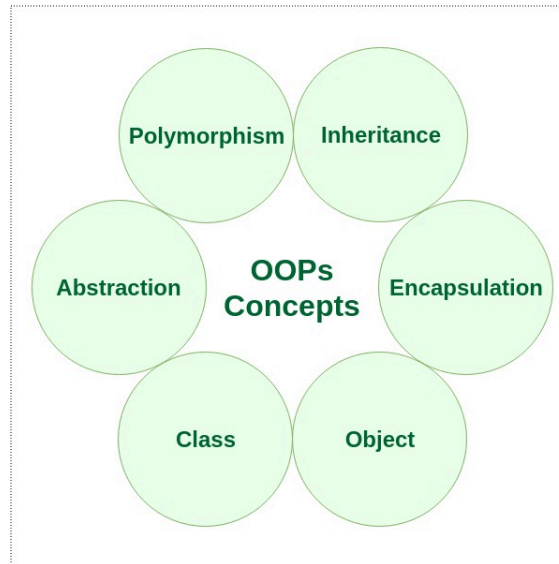
```
27    }
```

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.

```
1  // Class Method and Method Passing
2  import java.io.*;
3
4  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:

```
1  // abstract class
2  abstract class GFG {
3      // abstract methods declaration
4      abstract void add();
5      abstract void mul();
6      abstract void div();
7  }
```

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:

```
1  // Encapsulation using private modifier
2
3  // Employee class contains private data
4  // called employee id and employee name
5  class Employee {
6
7      private int empid;
8      private String ename;
9
10     // Setter methods
11     public void set_id(int empid) {
12         this.empid = empid;
13     }
14
15     public void set_name(String ename)
16     {
17         this.ename = ename;
18     }
19
20     // Getter methods
21     public int get_id() {
22         return empid;
23     }
24
25     public String get_name() {
26         return ename;
27     }
28 }
```

```
29
30  public class Geeks {
31
32      public static void main(String args[])
33      {
34          Employee e = new Employee();
35          e.set_id(78);
36          e.set_name("John");
37
38          System.out.println("Employee id: " +
39      e.get_id());
40          System.out.println("Employee Name: "
41      + e.get_name());
42      }
```

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:

```
1  //base class or parent class or super class
2  class A{
3      //parent class methods
4      void method1(){}
5      void method2(){}
6  }
7
8  //derived class or child class or base class
9  class B extends A{ //Inherits parent class methods
10     //child class methods
11     void method3(){}
12     void method4(){}
13 }
```

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:

```
1  sleep(1000) //millis
2  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:

```
1  // Java Program to Demonstrate
2  // Method Overloading and Overriding
3
4  // Parent Class
5  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 " +
15         a);
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.

[Comment](#)[More info](#)[Advertise with us](#)[Next Article](#)[Classes and Objects in Java](#)

Similar Reads

Java Tutorial

Java is one of the most popular and widely used programming languages. Used to develop mobile apps, desktop apps, web apps, web servers, games, and enterprise-level systems. Java was invented by James...

4 min read

Java Overview

Java Basics

Java Flow Control

Java Methods

Java Arrays

Java Strings

Java OOPs Concepts

Java OOP(Object Oriented Programming) Concepts

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 ar...

11 min read

Classes and Objects in Java

In Java, classes and objects are basic concepts of Object Oriented Programming (OOPs) that are used to represent real-world concepts and entities. The class represents a group of objects having similar propertie...

11 min read

Java Constructors

Java constructors or constructors in Java is a terminology used to construct something in our programs. A constructor in Java is a special method that is used to initialize objects. The constructor is called when an...

8 min read

Object Class in Java

Object class in Java is present in java.lang package. Every class in Java is directly or indirectly derived from the Object class. If a class does not extend any other class then it is a direct child class of the Java Object...

8 min read

Abstraction in Java

Abstraction in Java is the process of hiding the implementation details and only showing the essential functionality or features to the user. This helps simplify the system by focusing on what an object does rath...

9 min read

Encapsulation in Java

Encapsulation in Java is a fundamental OOP (object-oriented programming) principle that combines data and methods in a class. It allows implementation details to be hidden while exposing a public interface for...

7 min read

Inheritance in Java

Java, 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...

12 min read

Polymorphism in Java

The word 'polymorphism' means 'having many forms'. In Java, polymorphism refers to the ability of a message to be displayed in more than one form. This concept is a key feature of Object-Oriented...

6 min read

Method Overloading in Java

In Java, Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both...

9 min read

Overriding in Java

Overriding in Java occurs when a subclass implements a method which is already defined in the superclass or Base Class. The method in the subclass must have the same signature as in the superclass. It allows the...

13 min read

Java Packages

Packages in Java are a mechanism that encapsulates a group of classes, sub-packages, and interfaces. Packages are used for: Prevent naming conflicts by allowing classes with the same name to exist in differen...

7 min read

Java Interfaces

Java Collections

Java Exception Handling

Java Multithreading

Java File Handling

Java Streams and Lambda Expressions

Java IO

Java Synchronization

Java Regex

Java Networking

JDBC

Java Memory Allocation

Java Interview Questions

Java Practice Problems

Java Projects

**Corporate & Communications Address:**

A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305



Advertise with us

Company

About Us
Legal
Privacy Policy
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Web Technologies

HTML
CSS

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Python Tutorial

Python Programming Examples
Python Projects

[JavaScript](#)[TypeScript](#)[ReactJS](#)[NextJS](#)[Bootstrap](#)[Web Design](#)[Python Tkinter](#)[Web Scraping](#)[OpenCV Tutorial](#)[Python Interview Question](#)[Django](#)

Computer Science

[Operating Systems](#)[Computer Network](#)[Database Management System](#)[Software Engineering](#)[Digital Logic Design](#)[Engineering Maths](#)[Software Development](#)[Software Testing](#)

System Design

[High Level Design](#)[Low Level Design](#)[UML Diagrams](#)[Interview Guide](#)[Design Patterns](#)[OOAD](#)[System Design Bootcamp](#)[Interview Questions](#)

School Subjects

[Mathematics](#)[Physics](#)[Chemistry](#)[Biology](#)[Social Science](#)[English Grammar](#)[Commerce](#)[World GK](#)

DevOps

[Git](#)[Linux](#)[AWS](#)[Docker](#)[Kubernetes](#)[Azure](#)[GCP](#)[DevOps Roadmap](#)

Interview Preparation

[Competitive Programming](#)[Top DS or Algo for CP](#)[Company-Wise Recruitment Process](#)[Company-Wise Preparation](#)[Aptitude Preparation](#)[Puzzles](#)

GeeksforGeeks Videos

[DSA](#)[Python](#)[Java](#)[C++](#)[Web Development](#)[Data Science](#)[CS Subjects](#)