CONTENTS

// TUTORIAL //

OOPS Concepts in Java - OOPS Concepts Example

Published on August 4, 2022

Java

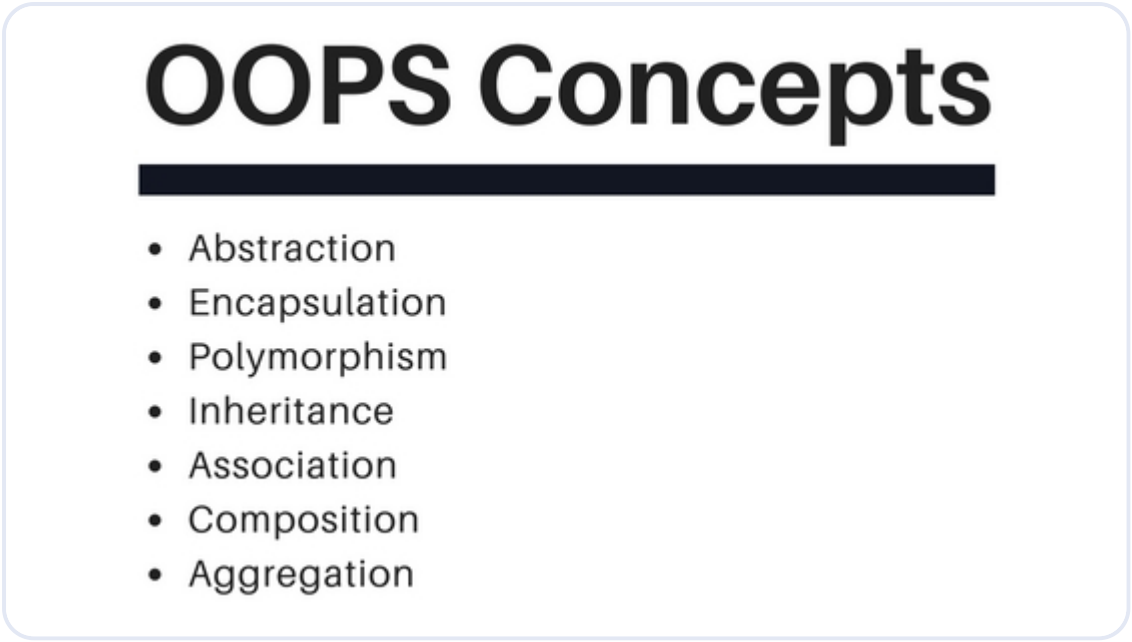
Pankaj

Object-Oriented Programming Concepts are very important for programming. Without having an idea about OOPS concepts, you will not be able to design systems in the object-oriented programming model.

What is Object-Oriented Programming Model?

The object-oriented programming model revolves around the concept of Objects. **What is an Object?** An object is an instance of a Class. It contains properties and functions. They are like real-world objects. For example, your car, house, laptop, etc. are all objects. They have some specific properties and methods to perform some action. **What is a Class?** The Class defines the blueprint of Objects. They define the properties and functionalities of the objects. For example, Laptop is a class and your laptop is an instance of it.

OOPS Concepts



Core OOPS concepts are:

1. Abstraction
2. Encapsulation
3. Polymorphism
4. Inheritance
5. Association
6. Aggregation
7. Composition

Let's look into these object-oriented programming concepts one by one. We will use Java programming language for code examples so that you know how to implement OOPS concepts in Java.

1. Abstraction

Abstraction is the concept of hiding the internal details and describing things in simple terms. For example, a method that adds two integers. The internal processing of the method is hidden from the outer world. There are many ways to achieve abstraction in object-oriented programmings, such as encapsulation and inheritance. A Java program is also a great example of abstraction. Here java takes care of converting simple statements to machine language and hides the inner implementation details from the outer world.

Further Reading: What is Abstraction in OOPS?

2. Encapsulation

Encapsulation is the technique used to implement abstraction in object-oriented programming. Encapsulation is used for access restriction to class members and methods. Access modifier keywords are used for encapsulation in object oriented

programming. For example, encapsulation in java is achieved using `private`, `protected` and `public` keywords.

3. Polymorphism

Polymorphism is the concept where an object behaves differently in different situations. There are two types of polymorphism - compile time polymorphism and runtime polymorphism. Compile-time polymorphism is achieved by method overloading. For example, we can have a class as below.

```
public class Circle {

	public void draw(){
		System.out.println("Drwaing circle with default color Black and diame
	}

	public void draw(int diameter){
		System.out.println("Drwaing circle with default color Black and diame
	}

	public void draw(int diameter, String color){
		System.out.println("Drwaing circle with color"+color+" and diameter"+
	}
}
```

Here we have multiple `draw` methods but they have different behavior. This is a case of method overloading because all the methods name is same and arguments are different. Here compiler will be able to identify the method to invoke at compile-time, hence it's called compile-time polymorphism. Runtime polymorphism is implemented when we have an "IS-A" relationship between objects. This is also called a method overriding because the subclass has to override the superclass method for runtime polymorphism. If we are working in terms of the superclass, the actual implementation class is decided at runtime. The compiler is not able to decide which class method will be invoked. This decision is done at runtime, hence the name as runtime polymorphism or dynamic method dispatch.

```
package com.journaldev.test;

public interface Shape {

	public void draw();
}
```

```
package com.journaldev.test;

public class Circle implements Shape{

	@Override
	public void draw(){
		System.out.println("Drwaing circle");
	}

}
```

```
package com.journaldev.test;

public class Square implements Shape {

	@Override
	public void draw() {
		System.out.println("Drawing Square");
	}

}
```

`Shape` is the superclass and there are two subclasses `Circle` and `Square`. Below is an example of runtime polymorphism.

```
Shape sh = new Circle();
sh.draw();

Shape sh1 = getShape(); //some third party logic to determine shape
sh1.draw();
```

In the above examples, java compiler doesn't know the actual implementation class of Shape that will be used at runtime, hence runtime polymorphism.

4. Inheritance

Inheritance is the object-oriented programming concept where an object is based on another object. Inheritance is the mechanism of code reuse. The object that is getting inherited is called the superclass and the object that inherits the superclass is called a subclass. We use `extends` keyword in java to implement inheritance. Below is a simple example of inheritance in java.

```
package com.journaldev.java.examples1;
```

```
class SuperClassA {

        public void foo(){
                System.out.println("SuperClassA");
        }

}

class SubClassB extends SuperClassA{

        public void bar(){
                System.out.println("SubClassB");
        }

}

public class Test {
        public static void main(String args[]){
                SubClassB a = new SubClassB();

                a.foo();
                a.bar();
        }
}
```

5. Association

Association is the OOPS concept to define the relationship between objects. The association defines the multiplicity between objects. For example Teacher and Student objects. There is a one-to-many relationship between a teacher and students. Similarly, a student can have a one-to-many relationship with teacher objects. However, both student and teacher objects are independent of each other.

Aggregation

Aggregation is a special type of association. In aggregation, objects have their own life cycle but there is ownership. Whenever we have “HAS-A” relationship between objects and ownership then it’s a case of aggregation.

6. Composition

The composition is a special case of aggregation. The composition is a more restrictive form of aggregation. When the contained object in “HAS-A” relationship can’t exist on its own, then it’s a case of composition. For example, House has-a Room. Here the room can’t exist without the house. Composition is said to be better than inheritance, read more at Composition vs Inheritance.

Further Reading: Composition in Java