Object-oriented programming (OOP) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) based on the concept of "[objects](https://en.wikipedia.org/wiki/Object_(computer_science))", which can contain [data](https://en.wikipedia.org/wiki/Data) and code: data in the form of [fields](https://en.wikipedia.org/wiki/Field_(computer_science)) (often known as attributes or properties), and code, in the form of procedures (often known as [methods](https://en.wikipedia.org/wiki/Method_(computer_science))).

**Simula** is considered the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.

**Smalltalk** is considered the first truly object-oriented programming language.

The popular object-oriented languages are [Java](https://www.javatpoint.com/java-tutorial), [C#](https://www.javatpoint.com/c-sharp-tutorial), [PHP](https://www.javatpoint.com/php-tutorial), [Python](https://www.javatpoint.com/python-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc.

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

OOPs (Object-Oriented Programming System)

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

* Coupling
* Cohesion
* Association
* Aggregation
* Composition



## Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.



**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

## Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

### **Inheritance**

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

This is a special feature of Object-Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.



### **Polymorphism**

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is **method overloading**. That’s when different meanings are implied by the code itself. The other form is **method overriding**. That’s when the different meanings are implied by the values of the supplied variables. See more on this below.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

#### **Abstraction**

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

Abstraction means using simple things to represent complexity. We all know how to turn the TV on, but we don’t need to know how it works in order to enjoy it. In Java, abstraction means simple things like **objects**, **classes**, and **variables** represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple tim es.

In Java, we use abstract class and interface to achieve abstraction.

### **Encapsulation**

**Encapsulation in Java** is a mechanism to wrap up variables(data) and methods(code) together as a single unit. It is the process of hiding information details and protecting data and behavior of the object. It is one of the four important OOP concepts. For example, a capsule, it is wrapped with different medicines.

To achieve a lesser degree of encapsulation in Java, you can use modifiers like "protected" or "public". With encapsulation, developers can change one part of the code easily without affecting other.

Data Hiding in Java is hiding the variables of a class from other classes. It can only be accessed through the method of their current class. It hides the implementation details from the users. But more than data hiding, it is meant for better management or grouping of related data.



A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

* declare class variables/attributes as private
* provide public get and set methods to access and update the value of a private variable

This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.

//Short Encapsulation

**class** Product {

**private** **int** id; // private means encapsulated

**private** String description; // private means encapsulated

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

}

**public** **class** EncapsulationExample {

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

Product p = **new** Product();

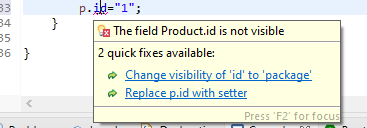
p.setId(1);

p.setDescription("IPhone");

System.***out***.println(p.getDescription());

}

}



Why Encapsulation?

* Better control of class attributes and methods
* Class attributes can be made **read-only** (if you only use the get method), or **write-only** (if you only use the set method)
* It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
* Flexible: the programmer can change one part of the code without affecting other parts
* With Java Encapsulation, you can hide (restrict access) to critical data members in your code, which improves security

Abstraction vs. Encapsulation

Often encapsulation is misunderstood with Abstraction. Lets study-

* Encapsulation is more about "How" to achieve a functionality
* Abstraction is more about "What" a class can do.

A simple example to understand this difference is a mobile phone. Where the complex logic in the circuit board is encapsulated in a touch screen, and the interface is provided to abstract it out.

### **Coupling**

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

### **Cohesion**

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

### **Association**

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

* One to One
* One to Many
* Many to One, and
* Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

### **Aggregation**

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a has-a relationship in Java. Like, inheritance represents the is-a relationship. It is another way to reuse objects.

### **Composition**

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

## Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

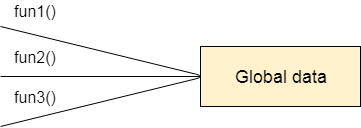


Figure: Data Representation in Procedure-Oriented Programming

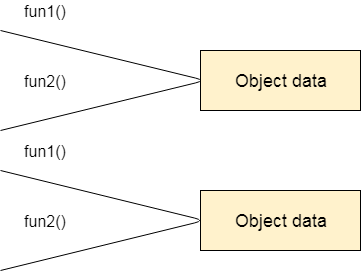


Figure: Data Representation in Object-Oriented Programming

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

## What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

Do You Know?

* Can we overload the main method?
* A Java Constructor returns a value but, what?
* Can we create a program without main method?
* What are the six ways to use this keyword?
* Why is multiple inheritance not supported in Java?
* Why use aggregation?
* Can we override the static method?
* What is the covariant return type?
* What are the three usages of Java super keyword?
* Why use instance initializer block?
* What is the usage of a blank final variable?
* What is a marker or tagged interface?
* What is runtime polymorphism or dynamic method dispatch?
* What is the difference between static and dynamic binding?
* How downcasting is possible in Java?
* What is the purpose of a private constructor?
* What is object cloning?

What will we learn in OOPs Concepts?

* Advantage of OOPs
* Naming Convention
* Object and class
* Method overloading
* Constructor
* static keyword
* this keyword with six usage
* Inheritance
* Aggregation
* Method Overriding
* Covariant Return Type
* super keyword
* Instance Initializer block
* final keyword
* Abstract class
* Interface
* Runtime Polymorphism
* Static and Dynamic Binding
* Downcasting with instanceof operator
* Package
* Access Modifiers
* Encapsulation
* Object Cloning

Next Topic[Naming Convention in Java](https://www.javatpoint.com/java-naming-conventions)