

Java OOPs Concepts

1. Object-Oriented Programming
2. Advantage of OOPs over Procedure-oriented programming language
3. Difference between Object-oriented and Object-based programming language.

Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

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.

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

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
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

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**.

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.

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.

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

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

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

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 unidirectional 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.
- 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.

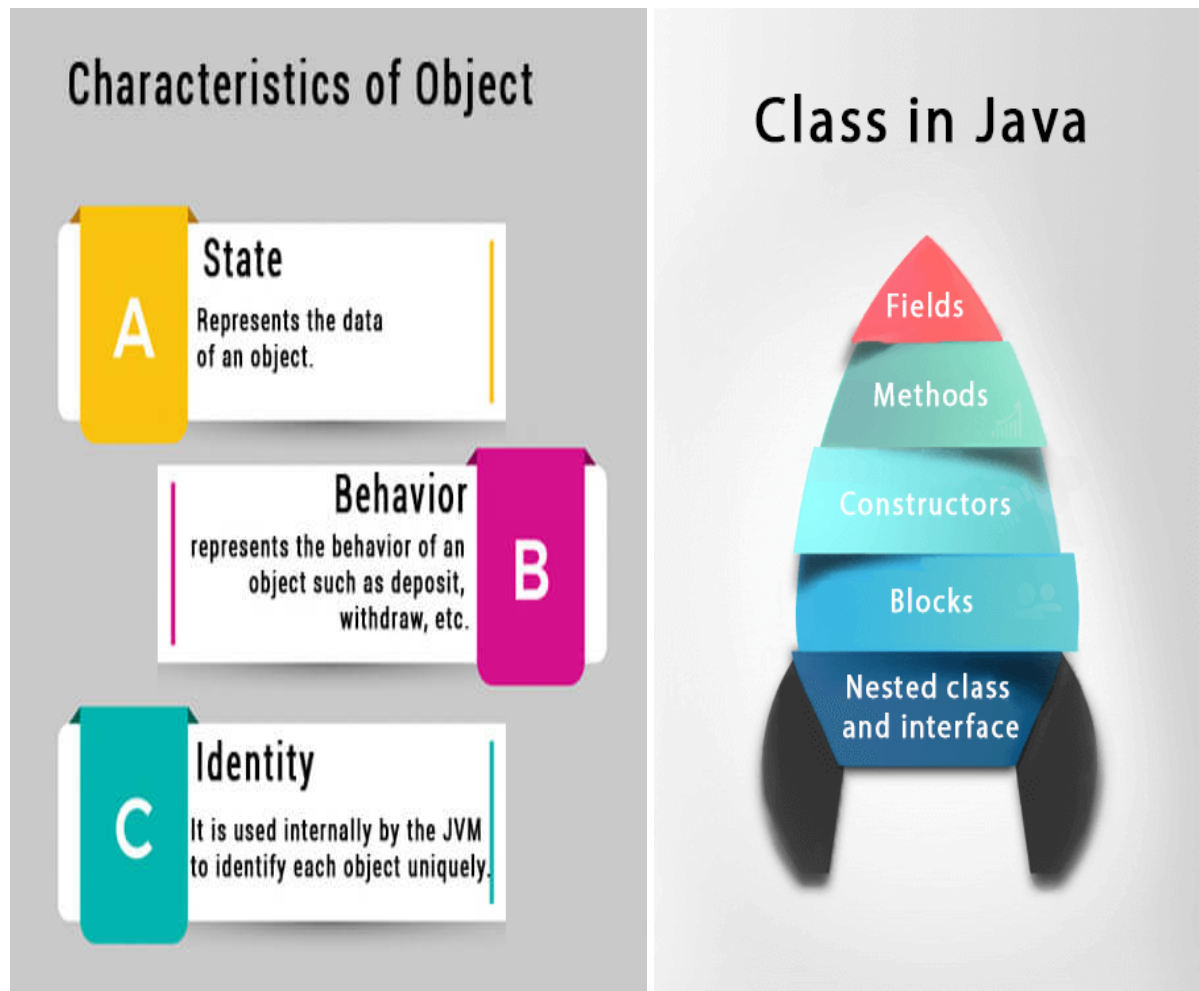
Naming Conventions of the Different Identifiers

The following table shows the popular conventions used for the different identifiers.

Identifiers Type	Naming Rules	Examples
Class	<p>It should start with the uppercase letter.</p> <p>It should be a noun such as Color, Button, System, Thread, etc.</p> <p>Use appropriate words, instead of acronyms.</p>	<pre>public class Employee { //code snippet }</pre>
Interface	<p>It should start with the uppercase letter.</p> <p>It should be an adjective such as Runnable, Remote, ActionListener.</p> <p>Use appropriate words, instead of acronyms.</p>	<pre>interface Printable { //code snippet }</pre>

Method	<p>It should start with lowercase letter.</p> <p>It should be a verb such as main(), print(), println().</p> <p>If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().</p>	<pre>class Employee { // method void draw() { //code snippet } }</pre>
Variable	<p>It should start with a lowercase letter such as id, name.</p> <p>It should not start with the special characters like & (ampersand), \$ (dollar), _ (underscore).</p> <p>If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.</p> <p>Avoid using one-character variables such as x, y, z.</p>	<pre>class Employee { // variable int id; //code snippet }</pre>

Package	<p>It should be a lowercase letter such as java, lang.</p> <p>If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.</p>	<pre>//package package com.javatpoint; class Employee { //code snippet }</pre>
Constant	<p>It should be in uppercase letters such as RED, YELLOW.</p> <p>If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.</p> <p>It may contain digits but not as the first letter.</p>	<pre>class Employee { //constant static final int MIN_AGE = 18; //code snippet }</pre>



Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in **Heap** memory area.