**OOPS Concept:**

**Object-Oriented Programming System (OOPs)** is a programming concept that works on the principles of abstraction, encapsulation, inheritance, and polymorphism. It allows users to create objects they want and create methods to handle those objects. The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

**1) Class**

The class is one of the Basic concepts of OOPs which is a group of similar entities. It is only a logical component and not the physical entity. Lets understand this one of the OOPs Concepts with example, if you had a class called “Expensive Cars” it could have objects like Mercedes, BMW, Toyota, etc. Its properties(data) can be price or speed of these cars. While the methods may be performed with these cars are driving, reverse, braking etc.

A class is a template or blueprint from which objects are created.

In object-oriented programming, a class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods). The user-defined objects are created using the class keyword.

**2) Object**

An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object is one of the Java OOPs concepts which contains both the data and the function, which operates on the data. For example – chair, bike, marker, pen, table, car, etc.

**3) Inheritance**

Inheritance is one of the Basic Concepts of OOPs in which one object acquires the properties and behaviors of the parent object. It’s creating a parent-child relationship between two classes. It offers robust and natural mechanism for organizing and structure of any software.

**4) Polymorphism**

Polymorphism refers to one of the OOPs concepts in Java which is the ability of a variable, object or function to take on multiple forms. For example, in English, the verb *run* has a different meaning if you use it with *a laptop*, *a foot race*, and *business*. Here, we understand the meaning of *run* based on the other words used along with it. The same also applied to Polymorphism.

**5) Abstraction**

Abstraction is one of the OOP Concepts in Java which is an act of representing essential features without including background details. It is a technique of creating a new data type that is suited for a specific application. Lets understand this one of the OOPs Concepts with example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.

For abstraction we use abstract class and Interface.

* + Interface methods are by default abstract and public
  + Interface attributes are by default public, static and final

**6) Encapsulation**

Encapsulation is one of the best Java OOPs concepts of wrapping the data and code. In this OOPs concept, the variables of a class are always hidden from other classes. It can only be accessed using the methods of their current class. For example – in school, a student cannot exist without a class.

**7) Association**

Association is a relationship between two objects. It is one of the OOP Concepts in Java which defines the diversity between objects. In this OOP concept, all objects have their separate lifecycle, and there is no owner. For example, many students can associate with one teacher while one student can also associate with multiple teachers.

**8) Aggregation**

In this technique, all objects have their separate lifecycle. However, there is ownership such that child object can’t belong to another parent object. For example consider class/objects department and teacher. Here, a single teacher can’t belong to multiple departments, but even if we delete the department, the teacher object will never be destroyed.

**9) Composition**

Composition is a specialized form of Aggregation. It is also called “death” relationship. Child objects do not have their lifecycle so when parent object deletes all child object will also delete automatically. For that, let’s take an example of House and rooms. Any house can have several rooms. One room can’t become part of two different houses. So, if you delete the house room will also be deleted.

*CLASSLOADER:*

The **Java ClassLoader** is a part of the [**Java Runtime Environment**](https://www.geeksforgeeks.org/differences-jdk-jre-jvm/) that dynamically loads Java classes into the [**Java Virtual Machine**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/). The Java run time system does not need to know about files and file systems because of classloaders.

[Java classes](https://www.geeksforgeeks.org/classes-objects-java/) aren’t loaded into memory all at once, but when required by an application. At this point, the **Java ClassLoader** is called by the **JRE** and these ClassLoaders load classes into memory dynamically.

A Java Classloader is of **three types**:

* + **BootStrap ClassLoader:** A Bootstrap Classloader is a Machine code which kickstarts the operation when the JVM calls it. It is not a java class. Its job is to load the first pure Java ClassLoader. Bootstrap ClassLoader loads classes from the location ***rt.jar***. Bootstrap ClassLoader doesn’t have any parent ClassLoaders. It is also called as the **Primodial ClassLoader**.
  + **Extension ClassLoader:** The Extension ClassLoader is a child of Bootstrap ClassLoader and loads the extensions of core java classes from the respective JDK Extension library. It loads files from ***jre/lib/ext*** directory or any other directory pointed by the system property ***java.ext.dirs***.
  + **System ClassLoader:** An Application ClassLoader is also known as a System ClassLoader. It loads the Application type classes found in the environment variable ***CLASSPATH, -classpath or -cp command line option***. The Application ClassLoader is a child class of Extension ClassLoader.