**Object Oriented Programming**

Object-oriented programming System (OOPs) is a programming paradigm based on the concept of “objects” that contain data and methods. The primary purpose of object-oriented programming is to increase the flexibility and maintainability of programs. Object oriented programming brings together data and its behaviour(methods) in a single location(object) makes it easier to understand how a program works. Java is mature OOP language.

A feature of objects is that an object's own procedures can access and often modify the data fields of itself (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another.[1][2] OOP languages are diverse, but the most popular ones are class-based.

Significant object-oriented languages include Java, C++, C#, Python, R, PHP, Visual Basic.NET, JavaScript, Ruby, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, and Small-talk.

**OOP Core Features**

**Abstraction** is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it. In Java, abstraction is achieved using Abstract classes and interfaces.

**Encapsulation** in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

To achieve encapsulation in Java −

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

**Inheritance** can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

**extends** is the keyword used to inherit the properties of a class. In the example below class **Sub** inherits some of the properties from class **Super.**

class Super {

   .....

   .....

}

class Sub **extends** Super {

   .....

   .....

}

**Polymorphism** means to process objects differently based on their data type. In other words, it means, one method with multiple implementations, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)

This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.

polymorphism can be implemented in two ways:

* **Overloading** in simple words means more than one method having the same method name that behaves differently based on the arguments passed while calling the method. This called static because, which method to be invoked is decided at the time of compilation.
* **Overriding** means a derived class is implementing a method of its super class. The call to overriden method is resolved at runtime, thus called runtime polymorphism.

**Feature of OOP**

**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 examples:

* One country can have one president - one to one,
* president can have many cabinet ministers - one to many.
* Many cabinet ministers can have one prime minister - many to one
* Many cabinet ministers can have many departments - many to many.

Association can be unidirectional or bidirectional.

**Aggregation**

Aggregation is a narrower kind of association. It occurs when there’s a one-way (HAS-A) relationship between the two classes you associate through their objects. For example, every Passenger has a Car but a Car doesn’t necessarily have a Passenger. When you declare the Passenger class, you can create a field of the Car type that shows which car the passenger belongs to. Then, when you instantiate a new Passenger object, you can access the data stored in the related Car as well.

Aggregation in Java:

* One-directional association.
* Represents a HAS-A relationship between two classes.
* Only one class is dependent on the other.

**Composition**

Composition is a stricter form of aggregation. It occurs when the two classes you associate are mutually dependent on each other and can’t exist without each other. For example, take a Car and an Engine class. A Car cannot run without an Engine, while an Engine also can’t function without being built into a Car. This kind of relationship between objects is also called a PART-OF relationship.

Composition in Java:

* A restricted form of aggregation
* Represents a PART-OF relationship between two classes.
* Both classes are dependent on each other
* If one class ceases to exist, the other can’t survive alone.