**OOPS**

**1-Lets get the definitions first:**

**Class:** It is a blueprint or template for creating objects. It has a set of properties and behaviors that the objects will possess.

**Object:** An instance of a class that represents a particular entity or concept in the real world.

**Inheritance:** A mechanism in which a new class is derived from an existing class, inheriting all the properties and methods of the parent class, and allowing the new class to add its own unique properties and methods.

**Polymorphism:** Polymorphism is a concept in object-oriented programming that allows objects of different classes to be treated as if they are of the same class. This means that you can write code that can work with multiple types of objects, without having to know the specific details of each type.

**Encapsulation:** Encapsulation is the practice of hiding the implementation details of a class from the outside world and exposing a public interface through which other classes can interact with it. It is achieved by making the class's data members private and providing public getter and setter methods to access and modify them. Encapsulation helps in maintaining data integrity and preventing unauthorized access to the data.

**Abstraction:** The practice of focusing on the essential features of an object or system, and ignoring the irrelevant details, in order to simplify the design and implementation of the code.

**Aggregation**: Aggregation is a way of connecting two classes where one class contains a reference to another class, but the contained class can exist on its own or be part of multiple containing classes. For example, a department can exist independently of a university, but a university can have multiple departments.

**Cohesion:** Cohesion is how closely related the responsibilities of a class are. A class with high cohesion has a clear and focused responsibility, and its methods and properties are related to that responsibility. A class with low cohesion has a broad or unclear responsibility, and its methods and properties may not be well organized or related to each other. High cohesion is good for code because it makes it more understandable, maintainable, and reusable.

**Association:** Association is a connection or interaction between two classes. It can be one-way or two-way, and can be represented using different types of multiplicity, such as one-to-one, one-to-many, or many-to-many. For example, a teacher can have an association with a class, representing the fact that the teacher teaches the class. Associations can also have attributes, such as the grade that a student receives in a class.

**Composition:** Composition is a way of connecting two classes where the contained class is part of the containing class and cannot exist independently of it. For example, a car has an engine, but the engine cannot exist without the car.

**2-Lets Understand More About a Method/Function of a class:**

So a method is like some action or we can say a “Verb”. The actions that an object of a class or the class can perform is called as method in easy words.

**2.2-There are four types in a method:**

With Arguments and Return Type

Without Arguments but Return Type

With Arguments and No Return type

Without Arguments and Without Return Type.

**3-Now lets see for some code examples to understand the concepts better.**

**3.1-Class:**

public class Car {

private String model;

private int year;

public Car( String model, int year) {

this.model = model;

this.year = year;

}

public String getMake() {

return make;

}

public String getModel() {

return model;

}

public int getYear() {

return year;

}

}

**3.2-Object:**

Car myCar = new Car("Honda", "Civic", 2022);

System.out.println(myCar.getMake()); // Output: Honda

Inheritance:

public class Animal {

public void eat() {

System.out.println("I am eating.");

}

}

public class Dog extends Animal {

public void bark() {

System.out.println("Woof!");

}

}

Dog myDog = new Dog();

myDog.eat(); // Output: I am eating.

myDog.bark(); // Output: Woof!

**3.3-Polymorphism:**

There are two types of polymorphism in Java:

**Compile-time polymorphism or method overloading:** It occurs when there are multiple methods with the same name in a class, but with different parameters or argument types. The compiler determines which method to call based on the arguments passed during compile-time.

Example:

public class Calculator {

public int add(int x, int y) {

return x + y;

}

public double add(double x, double y) {

return x + y;

}

}

**Runtime polymorphism or method overriding:** It occurs when a subclass provides its own implementation of a method that is already defined in its superclass. The subclass method must have the same name, return type, and parameter list as the superclass method.

Example:

public class Animal {

public void makeSound() {

System.out.println("Animal is making a sound");

}

}

public class Dog extends Animal {

public void makeSound() {

System.out.println("Dog is barking");

}

}

**3.4-Abstraction:**

public abstract class Shape {

public abstract double getArea();

public abstract double getPerimeter();

}

public class Rectangle extends Shape {

private double length;

private double width;

public Rectangle(double length, double width) {

this.length = length;

this.width = width;

}

public double getArea() {

return length \* width;

}

public double getPerimeter() {

return 2 \* (length + width);

}

}

Note for above code: We also have a concrete class Rectangle that extends the Shape class and provides its own implementation of the abstract methods. The Rectangle class's implementation is hidden from the outside world, and it can be treated as a Shape object.

**3.5-Encapsulation:**

public class Person {

private String name;

private int age;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if(age >= 0) {

this.age = age;

}

}

}

Note for above code: In the above example, we have a Person class with private data members name and age. The access to these members is restricted to the class itself. We have also provided public getter and setter methods for these members, which can be used to access and modify them from outside the class.