Object Oriented Programming

Main Three Concepts of OOP

Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name "sales section".

Polymorphism

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

Inheritance

The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.

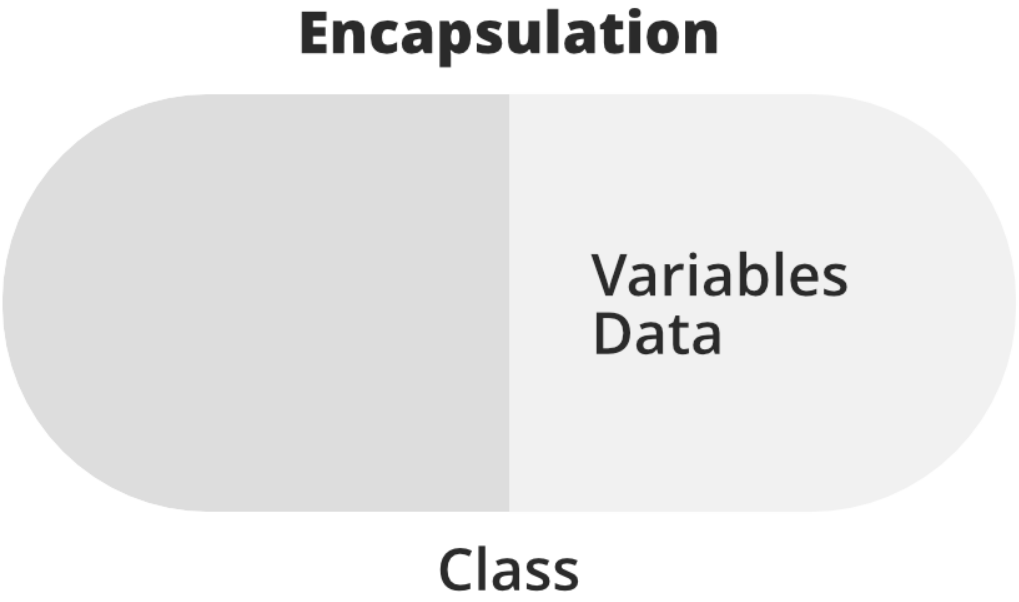

Figure 1: Encapsulation

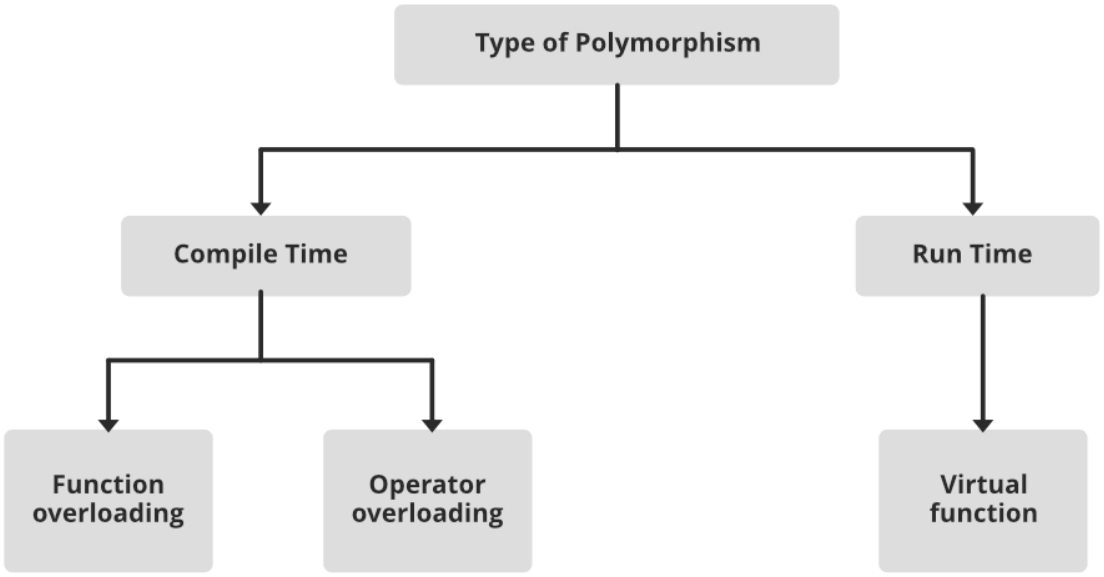

Figure 2: Polymorphism

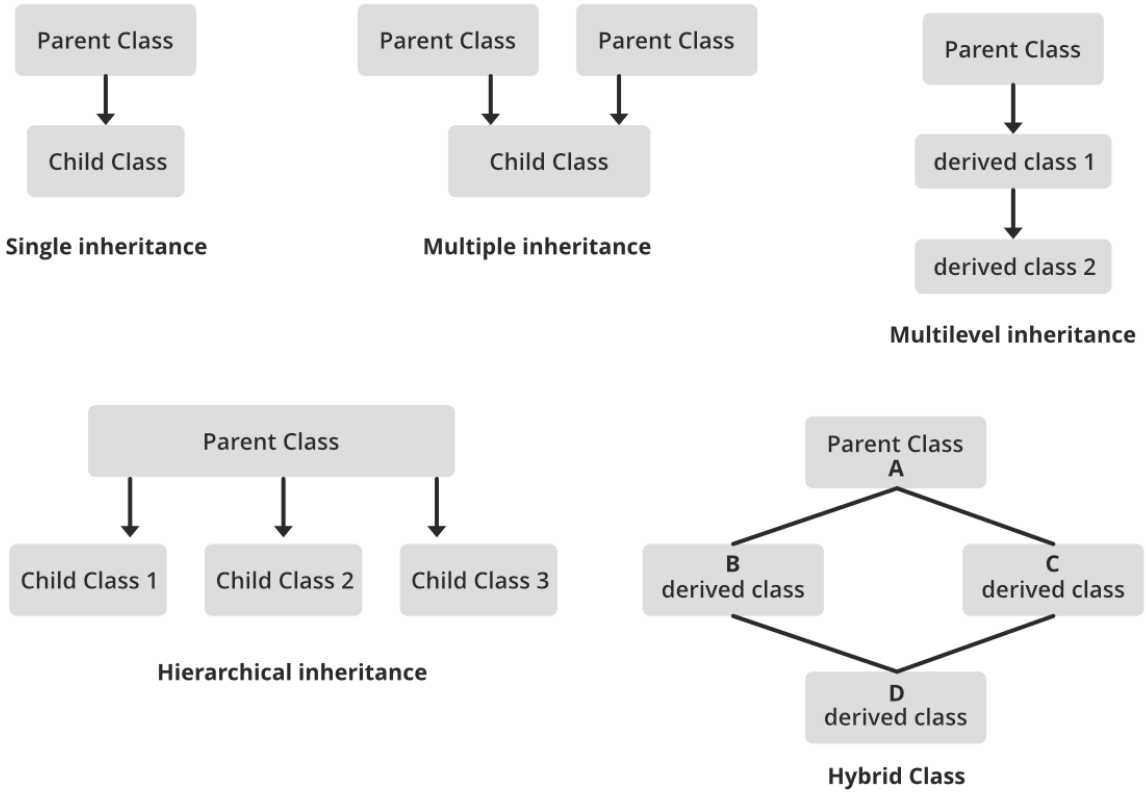

Figure 3: Inheritance

Single Inheritance When a class extends one class, that is a base class extends only one super class.

Multiple Inheritance When a class extends more than one class, that is the class has multiple super classes. No support in Java.

More Concepts

Polymorphic Variable A variable that can reference more than one type of object. Can be defined in a class that is able to reference objects of the class and objects of any of its descendants.

Overriding a method When a child class extends or overrides the implementation of its super class method.

Message Protocol of an Object Collections of an object's methods.

Nesting Class A nested class is a class within a class, it can be instance or static class.

Interface provides the definition of method interface without the implementation, which can be extended.

Class

A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

Object

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

For example "Dog" is a real-life Object, which has some characteristics like color, Breed, Bark, Sleep, and Eats.

Instantiation

It's the process where you create an instance of a class, which essentially means creating an object from a blueprint (class). When you instantiate a class, you're bringing it to life for use in your program. In object-oriented programming, every object must be instantiated before it can be used. This is because objects are instances of classes, and they must be created or "brought to life" from their class blueprints before they can do anything.

During instantiation, memory is allocated for the new object and its properties are set to their initial values. Then the constructor method of the class is called, if one exists. This method usually sets up the state of the object.

Abstract Data Type

- A type (or class) for objects whose behavior is defined by a set of values and a set of operations.
- It specifies what operations are to be performed but not how they will be implemented.
- Does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- Abstract gives an implementation-independent view.

Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as `a(int, int)` for two parameters, and `b(int, int, int)` for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

There are two ways to overload the method in Java

- By changing number of arguments
- By changing the data type

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

Interfaces and Abstract Classes

Interfaces	Abstract classes
Can only have final static variables. An interface can never change its own state.	Can have any kind of instance or static variables, mutable or immutable.
A class can implement multiple interfaces.	A class can extend only one abstract class.
Can be implemented with the implements keyword. An interface can also extend interfaces.	Can only be extended.
Can't have constructor.	Can have constructor.
Can have abstract methods.	Can have any kind of methods.

Overriding

Overriding is an object-oriented programming feature that enables a child class to provide different implementation for a method that is already defined and/or implemented in its parent class or one of its parent classes.

Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.

- Method overriding is used for runtime polymorphism
- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

Information Hiding

Information hiding is a programming principle that aims to prevent the direct modification of the data of a class. Also, it provides a strict guideline to access and modify the data of a class.

Additionally, it helps to hide design implementations from the client, especially design implementations that are likely to change.

```
class BookInformationHiding {
    private String author;
    private int isbn;
    private int id = 1;
```

```
    public BookInformationHiding(String author, int isbn) {
        setAuthor(author);
        setIsbn(isbn);
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public int getIsbn() {
        return isbn;
    }
    public void setIsbn(int isbn) {
        this.isbn = isbn;
    }
    public String getBookDetails() {
        return "author id: " + id + " author name: " + author + " ISBN: " + isbn;
    }
}
```

Here, we use the `private` access modifier to restrict access and altering of fields from an external class. Furthermore, we create getters and setters to set how the field can be accessed and modified.

In contrast to encapsulation without information hiding, where any access modifier can be used, information hiding is implemented through the use of a private access modifier. This restricts access to the fields within the class only.

The fields cannot be directly accessed from a client code, thus making it more robust.

Scope Visibility

Scope visibility refers to the accessibility or visibility of variables, functions, and other elements in a program, depending on the context in which they are defined. In object-oriented programming (OOP), scope visibility is controlled through the use of access modifiers, such as "public," "private," and "protected."

- **Public**: A public element can be accessed from anywhere in the program, both within the class and outside of it.
- **Private**: A private element can only be accessed within the class in which it is defined. It is not accessible to other classes, even if they inherit from the class.
- **Protected**: A protected element can only be accessed within the class and its subclasses.

Scoping

Class Scope

Variables declared inside a class but outside any method are class-level variables, also known as instance variables if they are non-static, or class variables if they are static. They are accessible to all methods in the class.

```
public class Example {
    private int instanceVariable;  // instance variable
    private static int classVariable;  // class variable

    public void method() {
        // instanceVariable and classVariable are accessible here
    }
}
```

Method Scope

Variables declared inside a method are local variables. They are only accessible within that method.

```
public void method() {
    int localVariable = 10;  // local variable
    // localVariable is accessible only within this method
}
```

Block Scope

Variables declared inside a block of code (e.g., within loops, if statements) are only accessible within that block.

```
public void method() {
    if (true) {
        int blockVariable = 20;  // block variable
        // blockVariable is accessible only within this block
    }
    // blockVariable is not accessible here
}
```

Parameter Scope

Method parameters are scoped to the method they are declared in.

```
public void method(int parameterVariable) {
    // parameterVariable is accessible only within this method
}
```

Members of a Class

TThe members of a class or an interface can be:

- **Fields (or Variables)**: These can be instance variables (non-static) or class variables (static).
- **Methods**: Functions defined within a class or an interface.
- **Constructors**: Special methods called when an object is instantiated.
- **Nested Types**: Classes or interfaces defined within another class.

Class Members

Consider the following example:

```
public class Bicycle {

    private int cadence;
    private int gear;
    private int speed;

    private int id;

    private static int numberOfBicycles = 0;


    public Bicycle(int startCadence,
                   int startSpeed,
                   int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;

        id = ++numberOfBicycles;
    }

    public int getID() {
        return id;
    }

    public static int getNumberOfBicycles() {
        return numberOfBicycles;
    }

    public int getCadence() {
        return cadence;
```

```
    }

    public void setCadence(int newValue) {
        cadence = newValue;
    }

    public int getGear(){
        return gear;
    }

    public void setGear(int newValue) {
        gear = newValue;
    }

    public int getSpeed() {
        return speed;
    }

    public void applyBrake(int decrement) {
        speed -= decrement;
    }

    public void speedUp(int increment) {
        speed += increment;
    }
}
```

Class Variables

When a number of objects are created from the same class blueprint, they each have their own distinct copies of instance variables. In the case of the `Bicycle` class, the instance variables are `cadence`, `gear`, and `speed`. Each `Bicycle` object has its own values for these variables, stored in different memory locations.

Sometimes, you want to have variables that are common to all objects. This is accomplished with the `static` modifier. Fields that have the `static` modifier in their declaration are called static fields or class variables. They are associated with the class, rather than with any object. Every instance of the class shares a class variable, which is in one fixed location in memory. Any object can change the value of a class variable, but class variables can also be manipulated without creating an instance of the class.

For example, suppose you want to create a number of `Bicycle` objects and assign each a serial number, beginning with 1 for the first object. This ID number is unique to each object and is therefore an instance variable. At the same time, you need a field to keep track of how many `Bicycle` objects have been created so that you know what ID to assign to the next one. Such a field is not related to any individual object, but to the class as a whole. For this

you need a class variable, `numberOfBicycles`.

Class Methods

A common use for static methods is to access static fields. For example, we could add a static method to the `Bicycle` class to access the `numberOfBicycles` static field.

Not all combinations of instance and class variables and methods are allowed:

- Instance methods can access instance variables and instance methods directly.
- Instance methods can access class variables and class methods directly.
- Class methods can access class variables and class methods directly.
- Class methods ***cannot*** access instance variables or instance methods directly—they must use an object reference. Also, class methods cannot use the `this` keyword as there is no instance for `this` to refer to.

Constants

The `static` modifier, in combination with the `final` modifier, is also used to define constants. The final modifier indicates that the value of this field cannot change.

For example, the following variable declaration defines a constant named `PI`, whose value is an approximation of π:

```
static final double PI = 3.141592653589793;
```

Constants defined in this way cannot be reassigned, and it is a compile-time error if your program tries to do so.