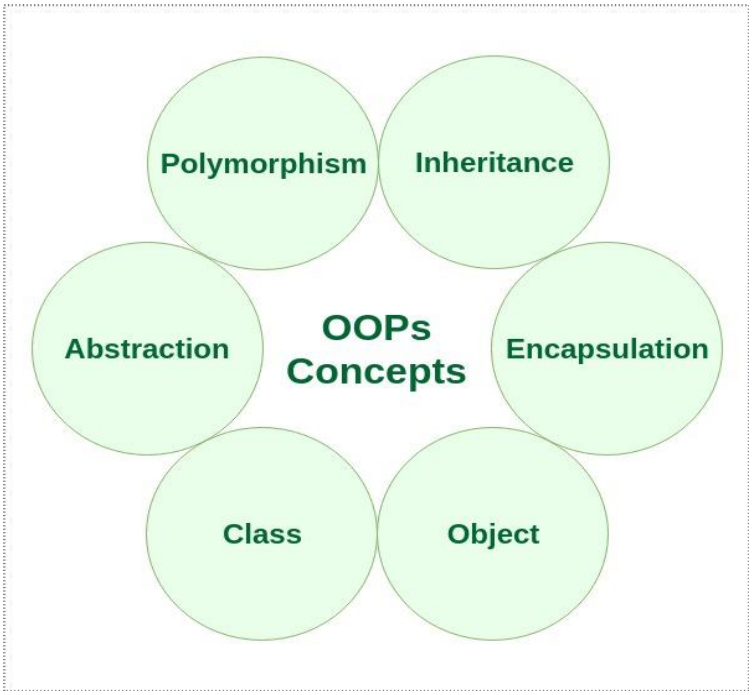
Object Oriented Programming (OOPs)

Object-Oriented Programming or Java OOPs concept refers to languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks you assign.

Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming. The main aim of OOPs is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.



4 pillars of OOPs which are as follows. But, let us start by learning about the different characteristics of an Object-Oriented Programming Language.

1. Class
2. Object
3. **Pillars of OOPs**
    - Abstraction
    - Encapsulation
    - Inheritance
    - Polymorphism
        - Compile-time polymorphism
        - Runtime polymorphism

Class :

A class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. Using classes, you can create multiple objects with the same behavior instead of writing their code multiple times. This includes classes for objects occurring more than once in your code.

In general, class declarations can include these components in order:

Modifiers: A class can be public or have default access (Refer to this for details).

Class name: The class name should begin with the initial letter capitalized by convention.

Superclass (if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

Body: The class body is surrounded by braces, { }.

Object :

An object is a basic unit of Object-Oriented Programming that represents real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. The objects are what perform your code, they are the part of your code visible to the viewer/user. An object mainly consists of:

State: It is represented by the attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.

Identity: It is a unique name given to an object that enables it to interact with other objects.

Method: A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping it, which is why they are considered time savers. In Java, every method must be part of some class, which is different from languages like C, C++, and Python.

Pillar 1: **Abstraction**

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or non-essential units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object, ignoring the irrelevant details. The properties and behaviors of an object differentiate it from other objects of similar type and also help in classifying/grouping the object.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the car speed or applying brakes will stop the car, but he does not know how on pressing the accelerator, the speed is actually increasing. He does not know about the inner mechanism of the car or the implementation of the accelerators, brakes etc. in the car. This is what abstraction is.

In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

The abstract method contains only method declaration but not implementation.

Exa :

```
//abstract class
abstract class Calc
{
        //abstract methods declaration
        abstract void add();
        abstract void mul();
        abstract void div();
}
```

Pillar 2: **Encapsulation**

It is defined as the wrapping up of data under a single unit. It is the mechanism that binds together the code and the data it manipulates. Another way to think about encapsulation is that it is a protective shield that prevents the data from being accessed by the code outside this shield.

Technically, in encapsulation, the variables or the data in a class is hidden from any other class and can be accessed only through any member function of the class in which they are declared.

In encapsulation, the data in a class is hidden from other classes, which is similar to what data-hiding does. So, the terms “encapsulation” and “data-hiding” are used interchangeably.

Encapsulation can be achieved by declaring all the variables in a class as private and writing public methods in the class to set and get the values of the variables.

Exa :

```
//Encapsulation using private modifier
//Employee class contains private data called employee id and employee name
class  Employee
{
        private int Emp_ID;
        private String Emp_Name;
}
```

Pillar 3: **Inheritance**

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features (fields and methods) of another class. We are achieving inheritance by using extends keyword. Inheritance is also known as "is-a" relationship.

Let us discuss some frequently used important terminologies:

Superclass: The class whose features are inherited is known as superclass (also known as base or parent class).

Subclass: The class that inherits the other class is known as subclass (also known as derived or extended or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

Reusability: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Exa :

```
//base class or parent class or super class
class A
{
        //parent class methods
        void method1(){}
        void method2(){}
}

//derived class or child class or base class
class B extends A          //Inherits parent class methods
{
         //child class methods
        void method3(){}
         void method4(){}
}
```

Pillar 4: **Polymorphism**

It refers to the ability of object-oriented programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities. The ability to appear in many forms is called polymorphism.

Polymorphism in Java is mainly of 2 types:

- Overloading
- Overriding

Exa :

```
// Java program to Demonstrate Polymorphism
// This class will contain 3 methods with same name, yet the program will compile & run successfully


public class Addition
{
        // Overloaded sum()

        // This sum takes two int parameters
        public int sum(int x, int y)
        {
                return (x + y);
        }

        // Overloaded sum().
        // This sum takes three int parameters
        public int sum(int x, int y, int z)
        {
                return (x + y + z);
        }

        // Overloaded sum().
        // This sum takes two double parameters
        public double sum(double x, double y)
        {
                return (x + y);
        }
```

```
    // Driver code
    public static void main(String args[])
    {
        Addition Obj = new Addition();

        System.out.println(Obj.sum(10, 20));
        System.out.println(Obj.sum(10, 20, 30));
        System.out.println(Obj.sum(10.5, 20.5));
    }
}
```

Advantage of OOPs over Procedure-oriented programming language

Object-oriented programming (OOP) offers several key advantages over procedural programming:

- **OOP promotes code reusability**: By using objects and classes, you can create reusable components, leading to less duplication and more efficient development.
- **OOP enhances code organization**: It provides a clear and logical structure, making the code easier to understand, maintain, and debug.
- **OOP supports the DRY (Don't Repeat Yourself) principle**: This principle encourages minimizing code repetition, leading to cleaner, more maintainable code. Common functionalities are placed in a single location and reused, reducing redundancy.
- **OOP enables faster development**: By reusing existing code and creating modular components, OOP allows for quicker and more efficient application development