

* OOPs (Object-Oriented Programming System)

Java is the object-oriented programming language as it supports the below concepts.

1. Class
2. Object
3. Package
4. Inheritance
5. Polymorphism
6. Encapsulation
7. Abstraction.

→ Inheritance → when you inherit from an existing class, you can reuse methods & fields of the parent class.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

In short,

Child class object will inherit the properties of parent class.

Java supports 4 different types of inheritance.

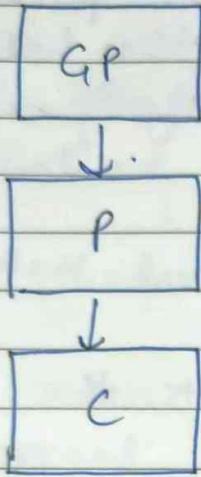
- Single inheritance.
- Multi-level inheritance
- Hierarchical inheritance
- Hybrid inheritance

* Single Inheritance : Inheriting the behaviour from one class to another class (Parent to child)

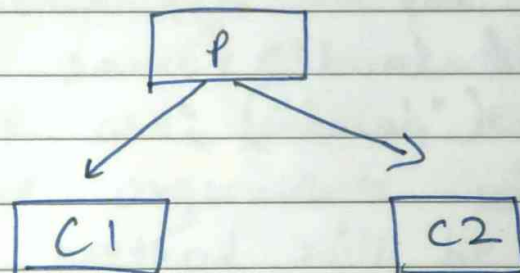
NOTE :

In inheritance, we can inherit the behaviour of one class to another class by using "Extends" keyword.

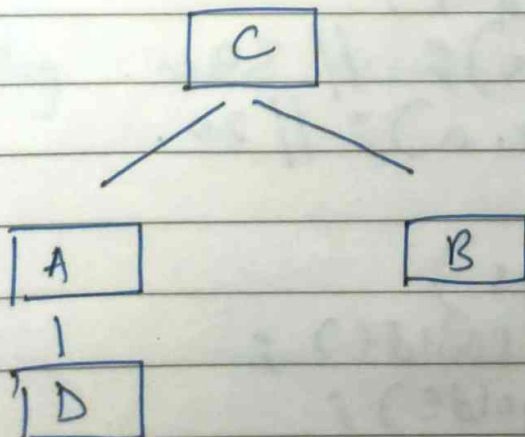
* Multilevel Inheritance : Inheriting properties from one class to another & into another class (Grandparents to Parents to children).



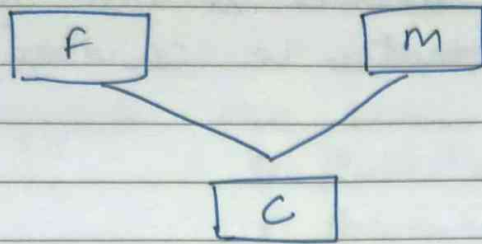
* Hierarchical Inheritance : Inheriting the behavior from one class to multiple class.



* Hybrid Inheritance : Inherit the behaviour from one class to another class into another class.



NOTE : Java does not support multiple inheritance



with the help of classes but can be possible with the help of interface.

⇒ this and super keyword in Java

this keyword is used to represent the current instance.

super keyword is used to access members of the parent class.

```
eg. class parent { int a = 20;
      public void land() {
        println("land");
      }
}
```

```
class child
{
  int a = 20;
  public void gold() {
    println("Gold");
    println(this.a); // 30
    println(super.a); // 20.
  }
}
```

```
psvm (String[] args)
{
  child c = new child();
  c.land(); c.gold();
}
```

→ Polymorphism → A concept by which we can perform a single action in different ways, where poly means "many" and morphs means "forms".

These are two types of polymorphism

1. Compile time polymorphism
2. Runtime polymorphism.

→ Method Overloading / Compile time Polymorphism

One class contains same method name with different arguments.

→ Method Overriding / Runtime Polymorphism

We are creating multiple methods with same name and same arguments.

In this we are creating the same method name with same arguments with inheritance & sub-class method will be overriding super class method.

NOTE : Java don't support constructor overriding.

Overriding can achieve only through inheritance.

→ Abstraction → Representing the essential features by hiding background details.

eg. A car is viewed as a car rather than its individual components.

So, in Java we can achieve this abstraction with the help of abstract class and abstract method.

* Abstract class and Abstract Methods

- An abstract class is a class that is declared with abstract keyword.
- An abstract method is a method that is declared without an implementation.
- Abstract class may or may not have all abstract methods. Some of them maybe concrete methods.
- Any class that contains one or more abstract methods must also be declared with abstract keyword.
- There can be no object of an abstract class.

Defined in the parent class & implemented in the child class.

NOTE, An abstract class can have both abstract & regular methods.

```
abstract class Animal {  
    public abstract void animalSound();  
    public void sleep() {  
        println("zzz");  
    }  
}
```

→ Encapsulation → Wrapping code and data together into a single unit.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now, we can use setter and getter methods to set and get the data in it.

As, in encapsulation, the data in a class is hidden from other classes, so it is also known as data binding.

⇒ Encapsulation can be achieved by : Declaring all the variables in the class as private and writing public methods in the class to set and get the value of variables.

* Get and Set Method

The get method returns the variable value, and the set method sets the value.

Syntax

get / set variable name (Uppercase first letter)

```
public class Person {  
    private String name; // private = restricted access
```

// Getter

```
    public String getName() {  
        return name; }  
}
```

// Setter

```
    public void setName (String newName) {  
        this.name = newName; }  
}
```