OOPs Concepts In Java

⇒ OOPS Overview

* OOPs means object oriented programming
* Here object means real world entity like Car, ATM, Bike etc.

Procedural Programming	OOPS
Program is divided into parts called functions.	Program is divided into objects
Doesn't provide a way to hide data, gives importance to function and data moves freely.	Objects provides data hiding, gives importance to data.
Overloading is not possible	Overloading is possible
Inheritence is not possible	Inheritence is possible
Code reusability does not present	Code reusability is present.
Eg:- Pascal, C, FORTRAN etc.	Eg:- Java, C#, Python, C++ etc.

⇒ Objects & Classes

* Object has 2 things :-
  - Properties or State
  - Behavior or Function

For Example :-

* Dog is an object because :-
  - Properties like : Age, Colour, breed etc.
  - Behavior like : Bark, sleep, eat etc.

* Car is an object because it has :-
  - Properties like Colour, Type, Brand, Weight etc.
  - Behavior like Apply brake, Drive, Increase speed etc.

* Class is a blueprint / skeleton of an object.
- To create an object, a Class is required.
- So, class provides the template or blueprint from which an object can be created
- From class, we can create multiple objects.
- To create a class, use keyword class :.

Eg:-

```
Class Student
{
    int age;
    string name;
    string address;
```

Data Variables

```
updateAddress ()
{
}
```
Data Method

```
getAge ()
{
    return age;
}
```
Data Method

Now let's create an object of type Student.

```
Student engStu = new Student ();
```

⇒ 1st Pillar of OOPs - Data Abstraction

* It hides the internal implementation and shows only essential functionality to the user.

* It can be achieved through Interface and abstract classes.

Example:-

- Car: we only show the BRAKE pedal, and if we press it, car speed will reduce. But HOW? That is ABSTRACTED from us.

- Cellphone: how call is made that is ABSTRACTED to us.

* Advantages of Abstraction:-

- It increases security & confidentiality

DEMO:-

```
Interface Car {

    public applyBrake();
    public incSpeed();
    public handbrake();
      :
      :
}

Class carImp Implements Car
{
    public applyBrake()
    {
        // step-1
        // step-2
        // step-3
           :
    }
}
```

So when user calls applyBrake(), internally it's invoking step-1, step-2 ... but all that is hidden from the user but ultimately car stops.

So this improves security as user is not aware of the internal functionality and only knows about the result.

⇒ 2nd Pillar of OOPS - DATA ENCAPSULATION

* Encapsulation bundles the data & code working on that data in a single unit.
* Also known as DATA-HIDING.
* Steps to achieve encapsulation
  - Declare variable of a class as private
  - Provide public getters & setters to modify & view the values of the variables

* Advantages of encapsulation :-
- Loosely coupled code
- Better access control & security.

DEMO:

```
class Dog
{
  private string Dog;

  string getColour()
{
  return this.colour;
}

  void setColour(string colour)
{
  this.colour = colour;
}
```

Now let's create an object of Dog type

```
Dog lab = new Dog();
lab.setColour("black");
lab.getColour; // will return black
```

So here we haven't given the access of the variable colour of class dog. Instead we did it with the help of the getter & setter which in turn have the access of variable.

⇒ 3rd Pillar of OOPS - INHERITENCE

* Capability of a class to inherit properties from their parent class.
* It can inherit both functions and variables so that we don't have to write them again in the child class.
* Can be achieved using extends keyword or through interface.
* Types of inheritence:-
  - Single inheritence
  - Multilevel inheritence
  - Hierarchical inheritence
  - Multiple inheritence (Not actually supported by Java due to diamond problem but through interface, we can solve the diamond problem.

* Advantages Of Inheritence
  - Code reusability
  - We can achieve polymorphism using inheritence

DEMO:-

Vehicle (Parent)

↓

Car (child)

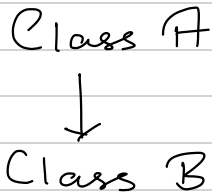
```
Class Vehicle
{
    boolean engine;
    boolean getEngine()
    {
      return this.engine;
    }
}

Class Car extends Vehicle
{
    string type;
    string getCarType()
    {
      return this.type;
    }
```

Now let's create an object of Car.

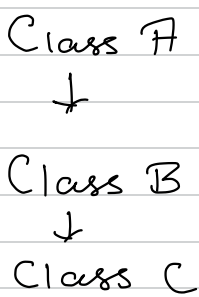
```
Car swift = new Car();
swift.getEngine();

Vehicle vehicle = new Vehicle();
vehicle.getCarType();   // Should not work
```

So, since swift is an object of Car which extends Vehicle hence it can call getEngine whereas vice versa isn't possible

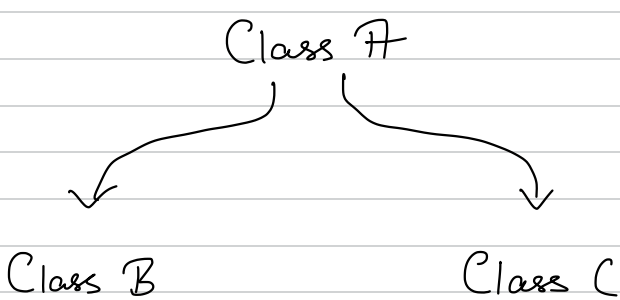
* Single



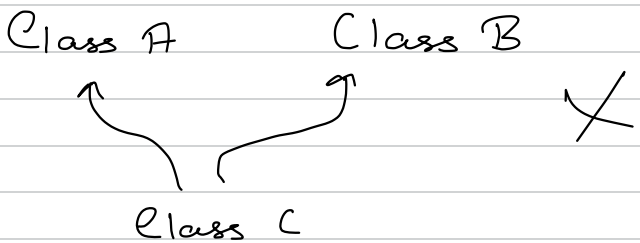
* Multilevel



* Hierarchical



* Multiple



This is not supported in Java due to diamond problem but there is a workaround for it using interfaces.

⇒ 4th Pillar of OOPS - POLYMORPHISM

* Poly means "Many" & morphism means "Form"
* A same method, behaves differently in different situation
* Example:
  - A person can be farmer, husband, employee etc.
  - Water can be liquid, solid or gas.

* Types of polymorphism:-
  - Compile Time / Static Polymorphism / Method Overloading
  - Run Time / Dynamic Polymorphism / Method Overriding

DEMO:

```
class Sum
{
    int doSum (int a, int b)
    {
      return a+b;
    }

    int doSum (int a, int b, int c)
    {
      return a+b+c;
    }
```

So this practice of creating methods with same name but different parameters is kla overloading.
So the method will be called based on the parameters.

```
Class A
{
  int getEngine ()
  {
     return 1;
  }
```

```
class B extends A
{
  int getEngine ()
  {
     return 2;
  }
```

Now let's create an object of class B

```
B obj = new B();
obj.getEngine(); // This 'll return 2
```

So which method to call is decided at runtime & this is kla method overriding.
So, in overriding, everything i.e. arguments, return type, method name is same.

* Is-a relationship
  - Achieved through Inheritence
  - Example: DOG is-a animal.
  - Inheritence form an is-a relation between its parent child classes

* Has-a relationship
  - Whenever, an object is used in other class, it's called HAS-A relationship.
  - Relation ship could be one-to-one, one-to-many, many-to-many
  - Example
    - School has students
    - Bike has engine
    - School has classes
  - Association: relationship between 2 different objects
    - Aggregation - Both objects can survive individually, means ending of one object will not end another object.
    - Composition - Ending of one object will end another object.