Java: OOP's Concept

By Saurav Saxena.

1. Ist PILLAR OF OOPS-DATA ABSTRACTION

- It hides the internal implementation and shows only essential functionality to user.
- It can be achieved through Interface and abstract classes.

Example:-

- Car We only show the break pedal and if we press it . Car speed will reduce. But How?? That is ABSTRACTION to us.
- Cellphone How call made that is Abstracted to us.

Advantages of abstraction:

- Increases security and confidentiality.
- Only Show Needed data to client.
- Simplify Our Client Code.

```
interface Car{
   public void applyBreak();
}
class CarImple implements Car{
   public void applyBreak(){
      //apply break;
      //turn on Headlight;
   }
}
```

2. SECOND PILLAR OF OOP'S - DATA ENCAPSULATION

- Also known as DATA-HIDING
- Steps to achieve encapsulation.

Declare Variable of a class as a private.
Provide a public getters and setters to Mofify and view those data.

```
class dog{
  private String color;

  String getColor(){
   return this.color;
  }
  void setColor(String color){
    this.color=color;
  }
}
```

Advantage of Encapsulation:-

- Loosely coupled code (changes occurs only in current object context not affecting other objects of same class.)
- Better access control and security

3. Third PILLAR OF OOP'S - INHERITANCE

- Capability of a class to inherit properties from their parent class.
- It can inherit both data and methods.
- It can be achieved using extends keyword or through interface.
- Types of Inheritance:-
- Single Inheritance.

  Multilevel Inheritance.
  Hierarchial Inheritance.
  Multiple Inheritance.

> *through interface we can resolve the diamond problem.*
> *In case of inheritance the code only provide declaration not implementation so after implementing those interface we are giving definition or implementation to that method of interface. that's how diamond problem get's resolved.*

Advantage :-

- Code reusability
- We can achieve Polymorphism using Inheritance.

```
class Vechile{
   boolean hasEngine;
   boolean getHasEngine(){
     return this.hasEngine;
   }
}

class Car extends Vechile{
  String carType;
  String getCarType(){
    return this.carType;
  }
}
```

```
Car swift = new Car();
swift.getHasEngine();//parent method accesable
Vechile v = new Vechile();
v.getCarType()//form parent child method is not accessable.
```

4. Forth PILLAR OF OOP'S - POLYMORPHISM

POLY->MANY
MOERPHIYSM->FORM

- A sample methods behaves differently in different situation.
- EXAMPLE:

A person can be father, husband, employee etc.
Water can be liquid , solid and gas etc,

Types of Polymorphism:

- Compile time/static polymorphism/Method overloading.
- Run Time/Dynamic Polymorphism / Method Overriding.

```
//Method overloading Examples
class Sum{
  int doSum(int a, int b){
    return a+b;
  }
  float doSum(float a, float b){
    return a+b;
  }
  String doSum(int a ,int b){
    return String.valueOf(a+b);
  }
  int doSum(int a, int b,int c,int d){
    return a+b+c+d;
    }
  }
```

```
  public class Main{
     public static void main(String[] args){
      Sum s = new Sum();
      System.out.println(s.doSum(10,10,10));
      int sum1 = s.doSum(10,5);
      String sum2 = s.doSum(2,3);//not correct overloading
     }
  }
```

- In Method overloading return type it doesn't matter because it only check parameter while calling a function at runtime it doesn't check the return type of that function.
- In case of overloading return type get's ignore it is totally based up on Parameters of method.
- In case of method overriding method signature (parameter,return type and function name) should be same.
- Only body might change of overridden function.

OBJECTS RELATIONSHIPS

- Is-a Relationship

  □Achieved through inheritance.
  □Example Dog Is-a animal
  Inheritance form an is-a relation its parent child class.

- Has-a Relationship
- Whenever an Object used in other Class it's called Has-A relationship.
- Relationship could be one-one,one-many,many to many.
- **Example**:
- School Has Students.
- Bike Has Engine.
- School Has classes.
- Association :Relationship between 2 different Objects.

- Aggeration[WEAK RELATION] : Both Objects, can survive individually means ending of one object will not end the other object.
- **Example: School and student**
- if school get's destroyed students get's enroll to some other school.
- Composition[STRONG RELATION]: Ending of one objects will end another objects.
- **Example: School and room**
- if School get's destroyed room get's also destroyed.