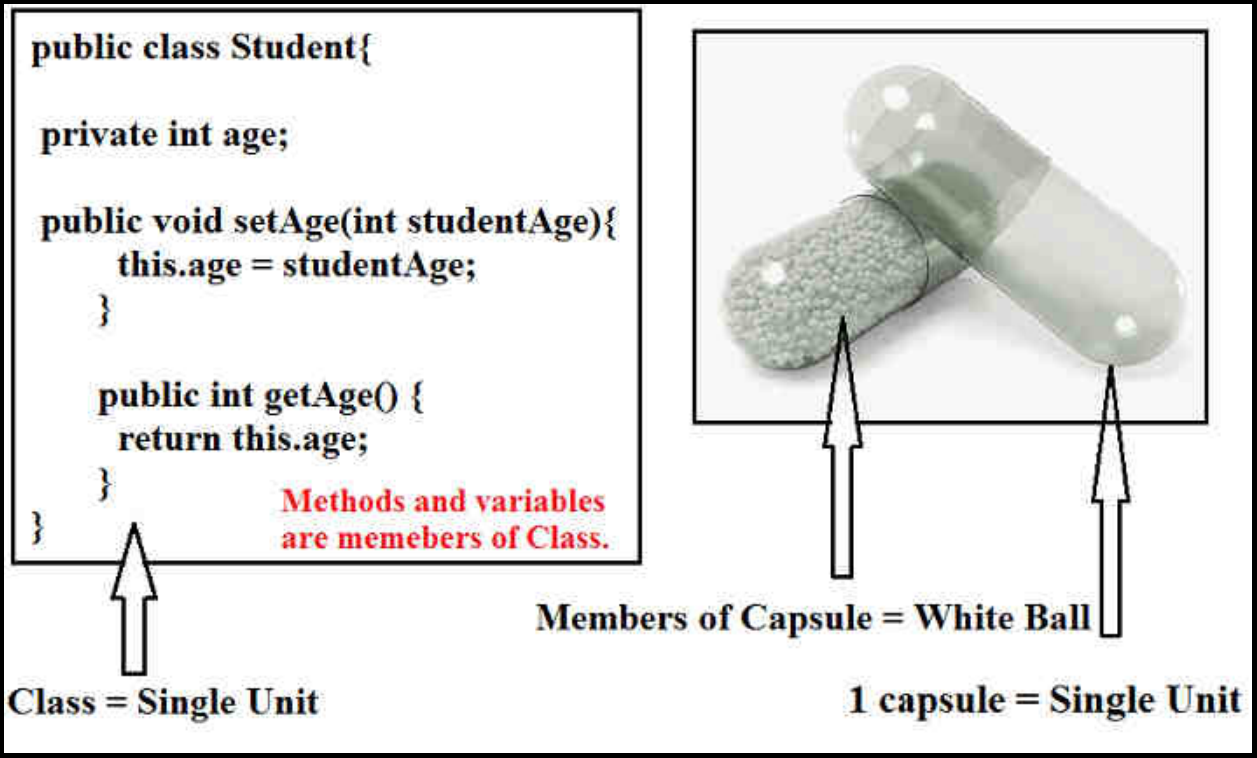
OOPS

ENCAPSULATION

- Encapsulation is mechanism through which we can binding the data member and member method in a single unit .for example of encapsulation is BANK because where be can perform multiple operation in one place.



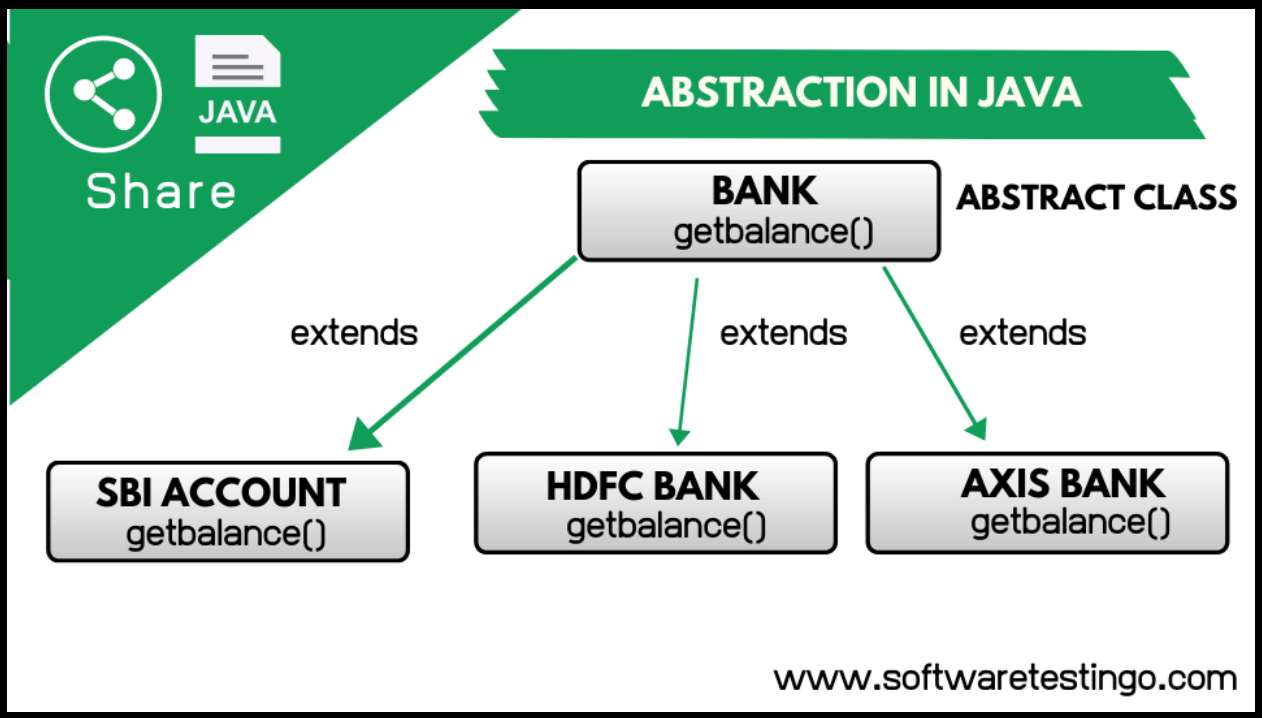
Program:

```
class Person {

    // private field
    private int age;

    // getter method
    public int getAge() {
        return age;
    }

    // setter method
```

```
    public void setAge(int age) {
        this.age = age;
    }
}

class Main {
    public static void main(String[] args) {

        // create an object of Person
        Person p1 = new Person();

        // change age using setter
        p1.setAge(24);

        // access age using getter
        System.out.println("My age is " + p1.getAge());
    }
}
OUTPUT:- My age is 24
```

ABSTRACTION

- Abstraction is nothing but handling the essential information and highlight the only set of services.



Abstract class

- If a class contain at least one abstract method called abstract method.
- We can not create object for abstract class
- It contains both abstract & non abstract method
- Whenever the action is common but implemention are different then we should use abstract method.

Interface

- Interface just like a class ,which contain only abstract method to achieve
- interface in java by the help of implements/interface keyword
- by default variable are public + static +final inside a interface
- by default method are public and abstract
- from jdk 1.8v onwards interface can have default & static method

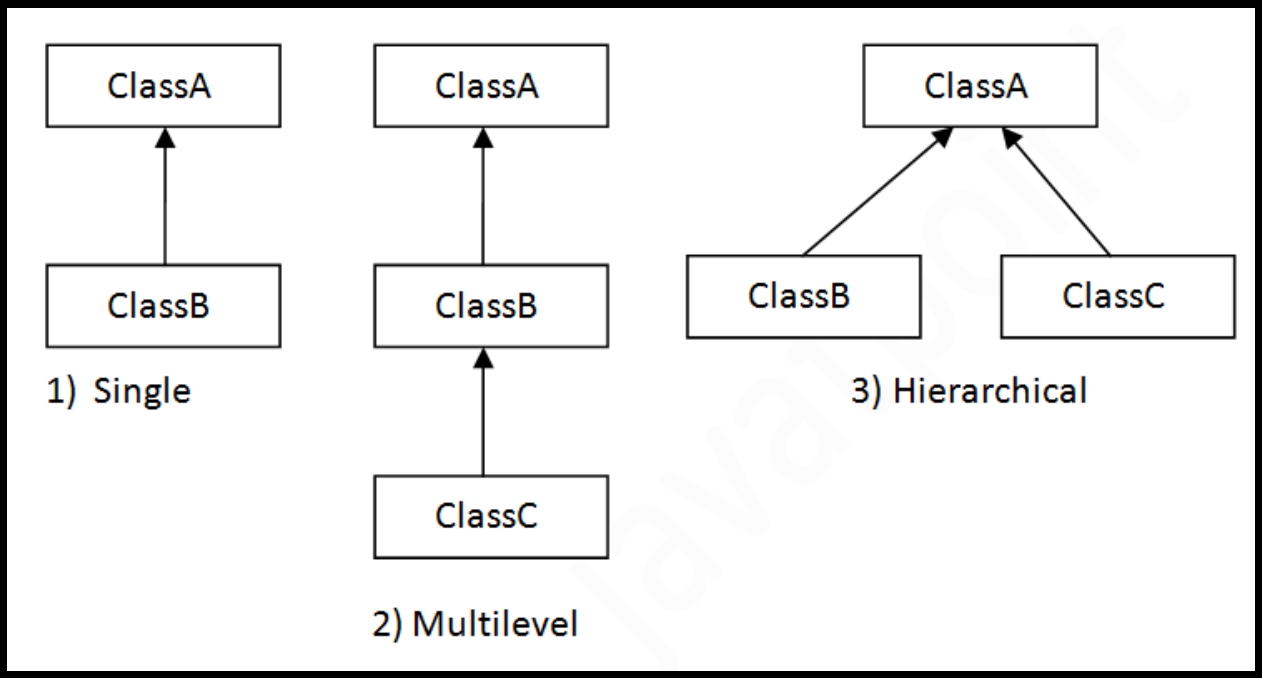
Program:

```
abstract class Bank {
    abstract int getRateOfInterest();
}

class SBI extends Bank {
    int getRateOfInterest() {
        return 7;
    }
}

class PNB extends Bank {
    int getRateOfInterest() {
        return 8;
    }
}

class TestBank {
    public static void main(String args[]) {
        Bank b;
        b = new SBI();
        System.out.println("Rate of Interest is: " + b.getRateOfInterest() + " %");
        b = new PNB();
        System.out.println("Rate of Interest is: " + b.getRateOfInterest() + " %");
    }
}
OUTPUT-
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

INHERITANCE

- When we construct a new class from existing class in such a way that the new class access all the features & properties of existing class called inheritance
- In java exstends keyword is used to perfrom inheritance
- It provides code reusability
- We can not access private member of class through inheritance
- A subclass contains all the features of super class so we should create the object of subclass
- Method overriding only possible through inheritance

Type:



1.Single Inheritance

- It also called a simple inheritance. Simple inheritance nothing but which contain only one super class and only one subclass is called simple inheritance

Synax:

```
class Bank {
 //code
}

class SBI extends Bank {
  //code
}
```

Program:

```
class Employee{
    float salary=40000;
```

```
   }
   class Programmer extends Employee{
    int bonus=10000;
    public static void main(String args[]){
      Programmer p=new Programmer();
      System.out.println("Programmer salary is:"+p.salary);
      System.out.println("Bonus of Programmer is:"+p.bonus);
   }
   }
OUTPut:-
Programmer salary is:40000.0
Bonus of Programmer is:10000
```

2.Multilevel Inheritance

- In multilevel inheritance be have only one super class and multiple sub classes called multiple inheritance

Synax:

```
class Animal{
 //code
}

class Dog extends Animal{
  //code
}
class BabyDog extends Dog{
  //code
}
```

Program

```
class Animal{
    void eat(){System.out.println("eating...");}
    }
    class Dog extends Animal{
    void bark(){System.out.println("barking...");}
    }
    class BabyDog extends Dog{
    void weep(){System.out.println("weeping...");}
    }
    class TestInheritance2{
    public static void main(String args[]){
    BabyDog d=new BabyDog();
    d.weep();
    d.bark();
    d.eat();
    }}
OUTPUT:-
weeping...
barking...
```

```
eating...
```

3.Hierarchical Inheritance

- A inheritance which contain only one super class and multiple sub class and all sub class directly extends super class called hierarchical inheritance

Synax:

```
class Animal{
 //code
}

class Dog extends Animal{
  //code
}
class Cat extends Animal{
  //code
}
```

Program:

```
class Animal{
    void eat(){System.out.println("eating...");}
    }
    class Dog extends Animal{
    void bark(){System.out.println("barking...");}
    }
    class Cat extends Animal{
    void meow(){System.out.println("meowing...");}
    }
    class TestInheritance3{
    public static void main(String args[]){
    Cat c=new Cat();
    c.meow();
    c.eat();
    //c.bark();//C.T.Error
    }}
OUTput:-
meowing...
eating...
```

*Multiple inheritance

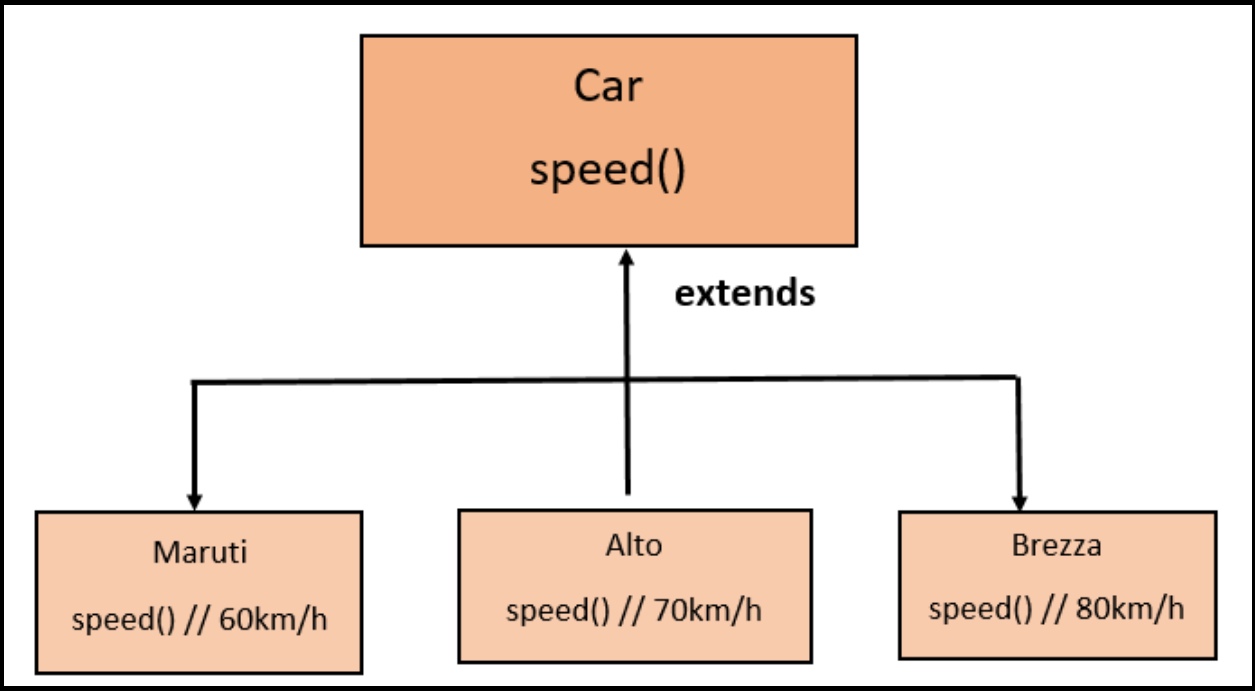
- We can achieve multiple inheritance through interfaces because interface contains only abstract method ,which implementation is provided by the sub classes

Program:

```java
class A
{
    public void execute()
    {
        System.out.println("Hi.. Executing From Class A");
    }
}
class B
{
    public void execute()
    {
        System.out.println("Hi.. Executing From Class B");
    }
}
class C extends A, B
{

}
public class Main
{
    public static void main(String[] args)
    {
        C obj = new C(); // creating object of class C
        obj.execute(); // execute() method is present in both class A and B
    }
}
```

POLYMORPHISM

- Polymorphism is the greek word whose meaning is “same object having different behaviour”



Type:

Compile time polymorphism

- A polymorphism which is exists at the time of compilation is called compile time. Early binding or static polymorphism

Eg. Method Overloading:

- Whenever a class contain more than one method with same name and different type of parameter called method overloading.

Syntax:

- Return-type method-name(para1);
- Return-type method-name(para2,para3);

Program:

```
class a {
    void add() {
        int a = 10, b = 20;
        int c = a + b;
        System.out.println(c);
    }

    void add(int x, int y) {
        int c = x + y;
        System.out.println(c);
    }

    void add(int x, double y) {
        double c = x + y;
        System.out.println(c);
    }

    public static void main(String[] args) {
        a obj = new a();
        obj.add();
        obj.add(30, 40);
        obj.add(50, 60.98);
    }
}
output:-
30
70
110.97999999999999
```

Runtime polymorphism

- A polymorphism which exists at the time of execution of program is called runtime polymorphism.

Eg.Method Overriding

- Whenever we writing method in super and sub classes in such a way that method name and parameter must be same called method overriding

Program:

```
class shape {
```

```
    void draw() {
        System.out.println("can't say shape type");
    }
}

class squre extends shape {
    @Override
    void draw() {
        super.draw();
        System.out.println("square shape");
    }
}

class demo {
    public static void main(String[] args) {
        squre obj = new squre();
        obj.draw();
    }
}
OUTPUT:
can't say shape type
square shape
```