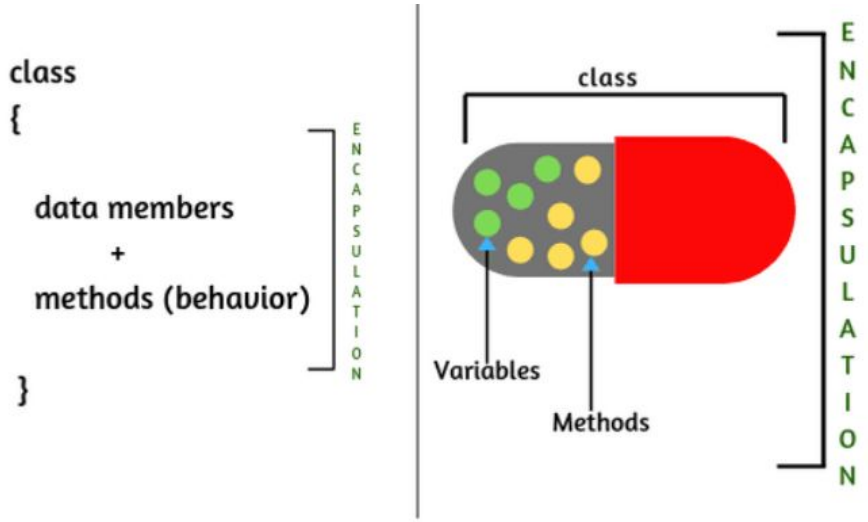
HNDIT3012 - OBJECT ORIENTED PROGRAMMING

LECTURE 06 – ENCAPSULATION & INHERITANCE

ENCAPSULATION

- It refers to the bundling of data with the methods that operate on that data
- Encapsulation in Java is the process by which data (variables) and the code that acts upon them (methods) are integrated as a single unit.
- By encapsulating a class's variables, other classes cannot access them, and only the methods of the class can access them

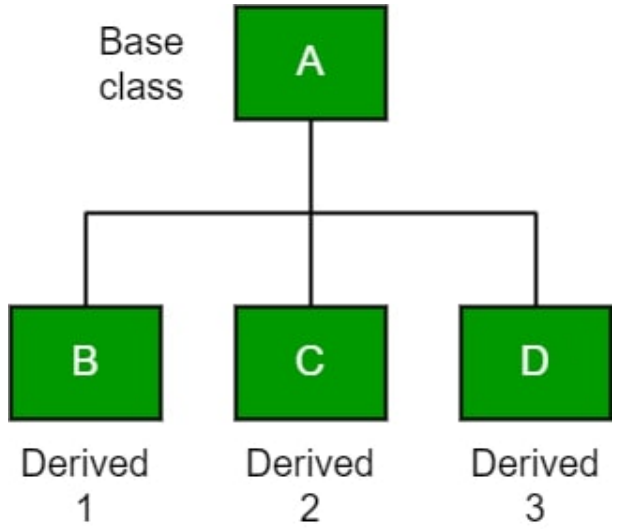


INHERITANCE

- A class that inherits from another class can reuse the methods and fields of that class.
- Java, Inheritance means creating new classes based on existing ones.
- it is possible to inherit attributes and methods from one class to another

"INHERITANCE CONCEPT"

- We group the into two categories:
- **subclass** (child/Drieved) - the class that inherits from another class
- **superclass** (parent/Base) - the class being inherited from



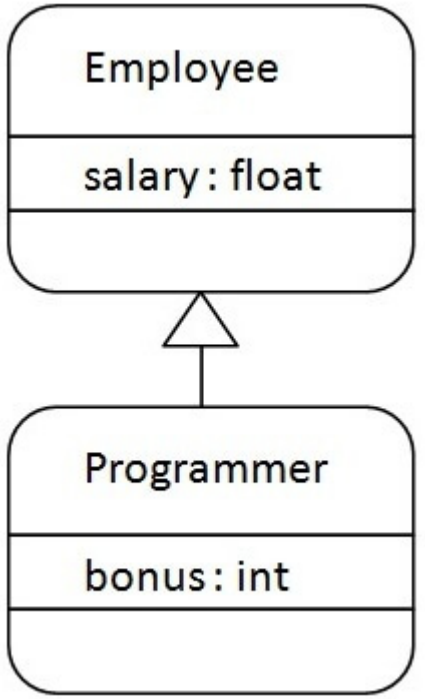
THE SYNTAX OF JAVA INHERITANCE

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

- The **extends keyword** indicates that you are making a new class that derives from an existing class.

- The meaning of "extends" is to increase the functionality.

EXAMPLE



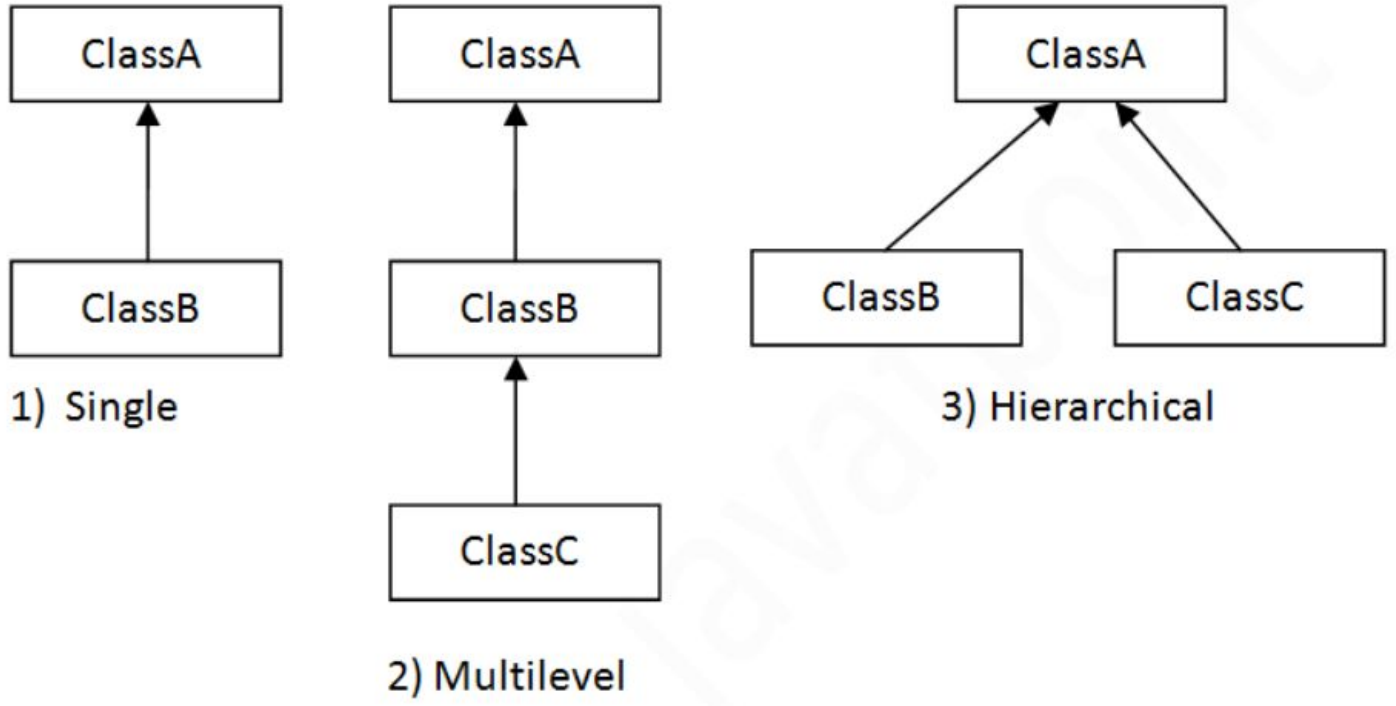

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;

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);

}
}
```

TYPES OF INHERITANCE IN JAVA

- On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.
- In java programming, multiple and hybrid inheritance is supported through interface only.



HYBRID INHERITANCE

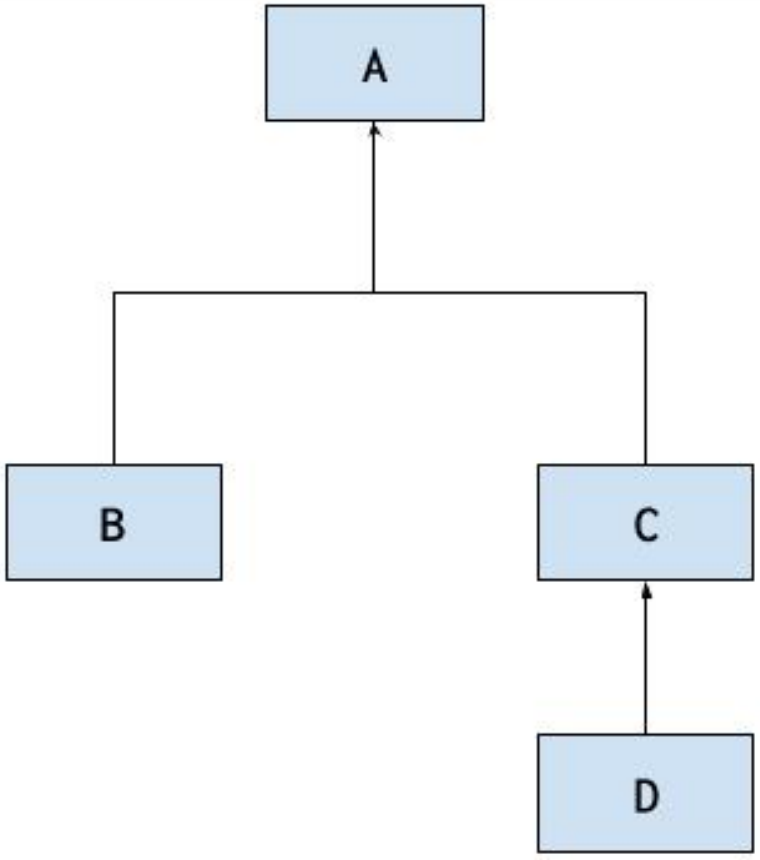
```
class A {

}
class B extends A {

}
class C extends A {

}
class D extends C {

}
```



SINGLE INHERITANCE EXAMPLE

- When a class inherits another class, it is known as a *single inheritance*.
- In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class Animal{
void eat(){
        System.out.println("eating...");
        }
}
class Dog extends Animal{
void bark(){
        System.out.println("barking...");
        }
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}
}
```

MULTILEVEL INHERITANCE EXAMPLE

- When there is a chain of inheritance, it is known as *multilevel inheritance*.
  - example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```
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();
}
}
```

HIERARCHICAL INHERITANCE EXAMPLE

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

- In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
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();
}
```

BASE CLASS

```
public class Animal
{    private boolean vegetarian;
     private String eats;
     private int noOfLegs;

 public Animal(boolean veg, String food, int legs) {
     this.vegetarian = veg;
     this.eats = food;
     this.noOfLegs = legs;
}
 public boolean isVegetarian()  {
     return vegetarian;
  }
 public void setVegetarian(boolean vegetarian)  {
     this.vegetarian = vegetarian;
}
 public String getEats() {
     return eats;
     }
 public void setEats(String eats)  {
     this.eats = eats;
     }
 public int getNoOfLegs()  {
     return noOfLegs;
}
 public void setNoOfLegs(int noOfLegs) {
     this.noOfLegs = noOfLegs;
     }
}
```

CHILD CLASS

```
public class Cat extends Animal{
private String color;
public Cat(boolean veg, String food, int legs) {
     super(veg, food, legs);
     this.color="White";
}
public Cat(boolean veg, String food, int legs, String color) {
     super(veg, food, legs);
     this.color=color;
}
public String getColor() {
     return color;
}
public void setColor(String color) {
     this.color = color;
}
```

MAIN CLASS

```
public class AnimalUse {
public static void main(String[] args)
{
Cat cat = new Cat(false, "milk", 4, "black");
System.out.println("Cat is Vegetarian?"+ cat.isVegetarian());
System.out.println("Cat eats " + cat.getEats());
System.out.println("Cat has " + cat.getNoOfLegs()+ " legs.");
System.out.println("Cat color is "+ cat.getColor());
}
}
```

SUPER KEYWORD IN JAVA

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

- Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

USAGE OF JAVA SUPER KEYWORD

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

SUPER KEY WORD

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class Test1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}
}
```

SUPER ☐ INVOKE PARENT CLASS METHOD

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
}
class Test2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

SUPER □ INVOKE PARENT CLASS CONSTRUCTOR

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

CURRENT CLASS INSTANCE VARIABLE

```
class Student{
    int rollno;
    String name;
    float fee;

Student(int rollno,String name,float fee){
    rollno=rollno;
    name=name;
    fee=fee;
}
void display(){
    System.out.println(rollno+" "+name+" "+fee);
}
}
```

```
class TestThis1{

public static void main(String args[]){
    Student s1=new Student(111,"ankit",5000f);

    Student s2=new Student(112,"sumit",6000f);

    s1.display();

    s2.display();
}

}
```

INVOKE CURRENT CLASS METHOD

```
class A{


void m(){
    System.out.println("hello m");
    }


void n(){
    System.out.println("hello n");
    //m();//same as this.m()
    this.m();
    }
}
```

```
class TestThis4{

public static void main(String args[]){
    A a=new A();
    a.n();
    }

}
```