Encapsulation,Inheritance types and examples, super keyword and super() method

-K.L.Madhavi

Encapsulation

- Encapsulation is one of the key features of object-oriented programming. Encapsulation refers to the bundling of fields and methods inside a single class.
- It prevents outer classes from accessing and changing fields and methods of a class. This also helps to achieve **data hiding**.
- In Java, encapsulation helps us to keep related fields and methods together, which makes our code cleaner and easy to read.
- The **Java Bean** class is the example of a fully encapsulated class.
- It helps to control the values of our data fields. For example,

```
class Person {
        private int age;
        public void setAge(int age) {
        if (age >= 0) {
        this.age = age;
        }
    }
}
```

Continue…

- The getter and setter methods provide **read-only** or **write-only** access to our class fields. For example,

  getName() // provides read-only access

  setName() //provides write-only access

- We can also achieve data hiding using encapsulation.Data hiding is a way of restricting the access of our data members by hiding the implementation details.
- 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 values of variables.
- It is more defined with the setter and getter method.
- **Advantages**:Data Hiding, Reusability,Increased Flexibility,Testing code is easy.

Continue…

- We can use access modifiers to achieve data hiding in encapsulation. For example,

```
Class Human
{
        private int age;
        private String name;

        public int getAge()
        {
             return age;
        }
        public void setAge(int  a)
        {
              age=a;
        }
        public String getName()
        {
              return name;
         }
Public void setName(String n)
{
        name=n;
}

Public class Demo
{
      public static void main(String args[])
      {
            Human obj=new Human();
            obj.setAge(22);
            obj.setName(”madhu”);
            System.out.println(obj.getName());
            System.out.println(obj.getAge());
        }
}
```

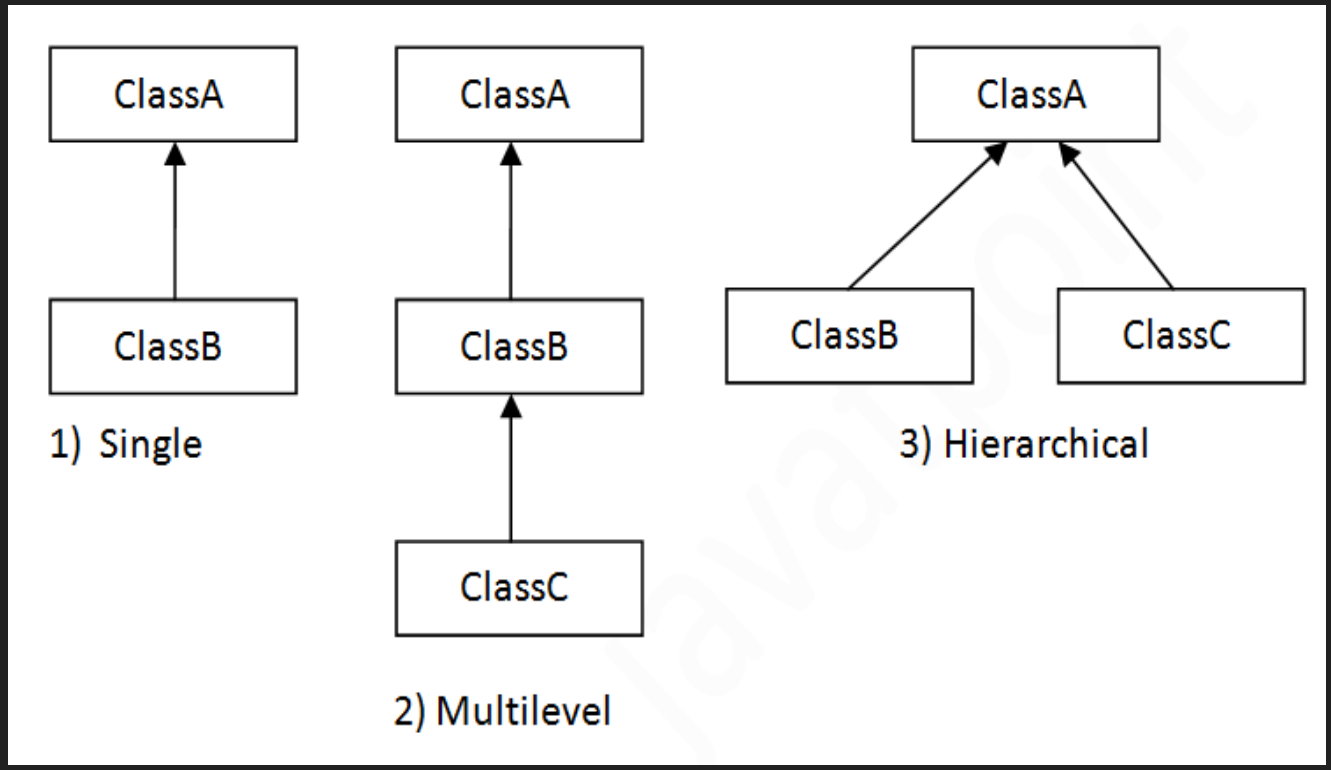
Inheritance

- Java, Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class.Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, we can add new fields and methods to your current class as well.
- **Syntax:**

```
class DerivedClass extends BaseClass
  {
      //methods and fields
   }
```

Types of inheritance in java

- On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.



Single-Level Inheritance

- 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.

Example Program:

```
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();
}}
```

Multi-Level Inheritance

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

Example 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();
}}
```

Hierarchical Inheritance

- 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.

Example 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();//Error
        }
}
```

Super Keyword

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

Usuage of super keyword:

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.

1) super is used to refer immediate parent class instance variable

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
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 TestSuper1{
    public static void main(String args[]){
    Dog d=new Dog();
    d.printColor();
}}
```

2.super can be used to invoke immediate parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
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 TestSuper2{
         public static void main(String args[]){
         Dog d=new Dog();
         d.work();
}}
```

Super() method

super() can be used to invoke immediate parent class constructor.

Example Program:

```
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();
}}
```