Object Oriented Programming System (OOPS)

Learn Java by Vaibhav Sir

Contents **Page No.**

What is Object?

- In the field of java each and everything is considered as object.
- Object is a copy of class or instance class which has state and behavior
- State Means Variable (Data Member)
- Behavior Means Method (Member Function)

Characteristics of Object:

1. **State (What it has?)**
2. **Behavior (What it can do?)**

E.g.	Marker
State: -	Color, Size, Weight, price
Behavior: -	Write Through

OOPS Concept provides 5 Important Principles.

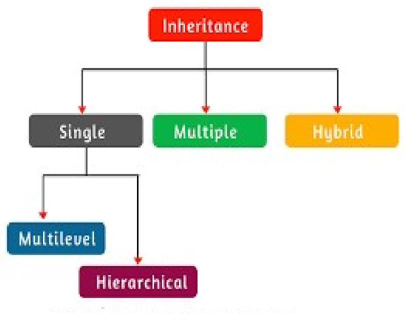
1. **Inheritance**
2. **Polymorphism**
3. **Encapsulation**
4. **Interface**
5. **Abstraction**

Inheritance:

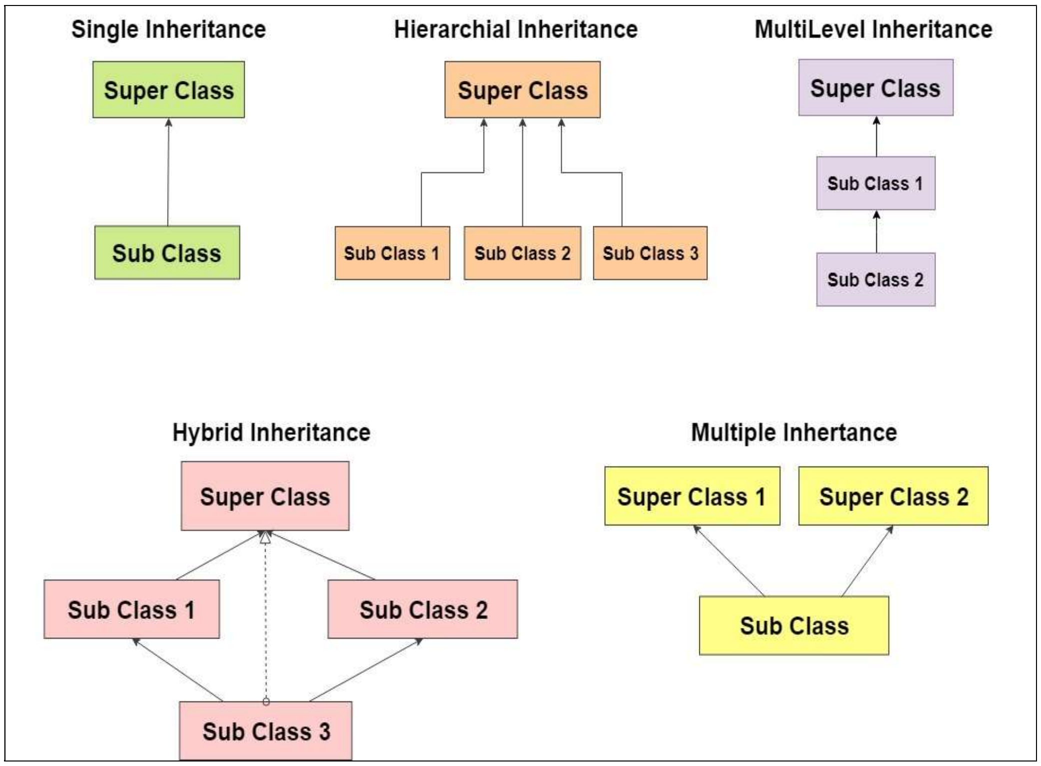
- It is one of the Oops principle where one class acquires properties of another class with the help of 'extends' keywords is called Inheritance.
- The class from where properties are acquiring/inheriting is called super/base/parent class.
- The class to where properties are inherited/delivered is called sub/child class.
- Inheritance takes place between 2 or more than 2 classes.

Inheritance is classified into 4 types:

1. **Single-level Inheritance**
2. **Multi-level Inheritance**
3. **Multiple Inheritance**
4. **Hierarchical Inheritance**

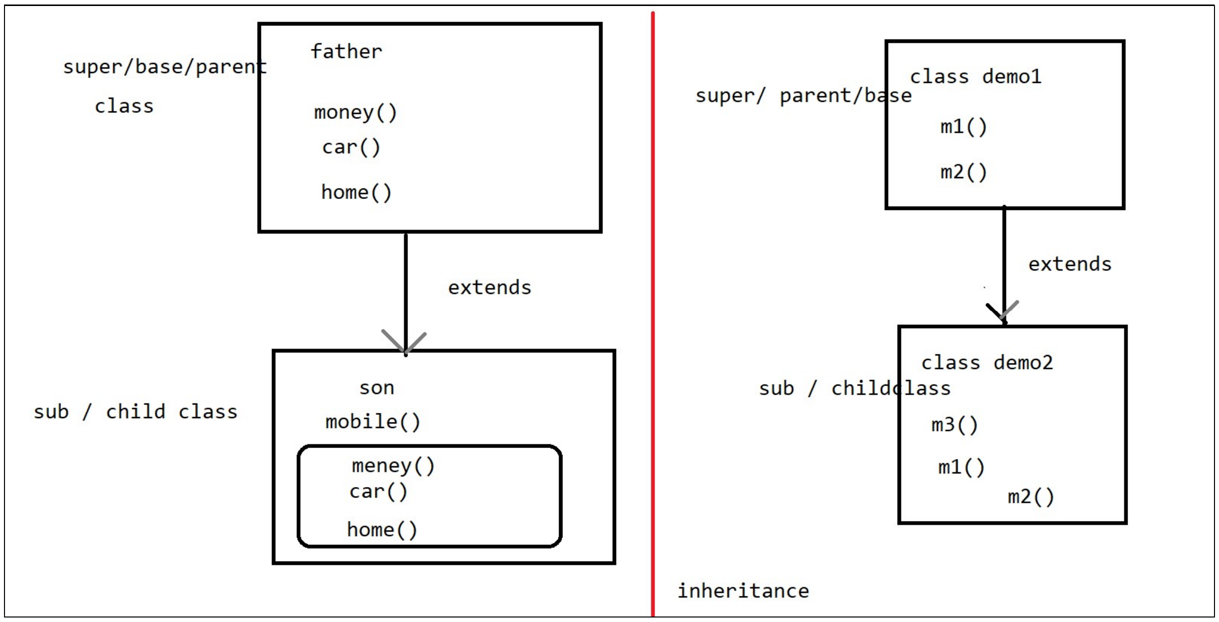


Classification of Inheritance in Java



1. Single-level Inheritance:

- It is an operation where inheritance takes place between 2 classes.
- To perform single-level inheritance only 2 classes are mandatory.
- If only one base class is used to derive only one subclass, then it is referred as single-level inheritance **or** **if only one sub class acquires property of one superclass, then it is refereed as single level inheritance.**



Example: Single-level Inheritance **(Class:1) (Parent/Base/Super Class)**

```
package Inheritance;                              //parent / base / super
public class father {

      public void money()
      {
               System.out.println("money");
      }
      public void car()
      {
               System.out.println("car");
      }
      public void home()
      {
               System.out.println("home");
      }
```

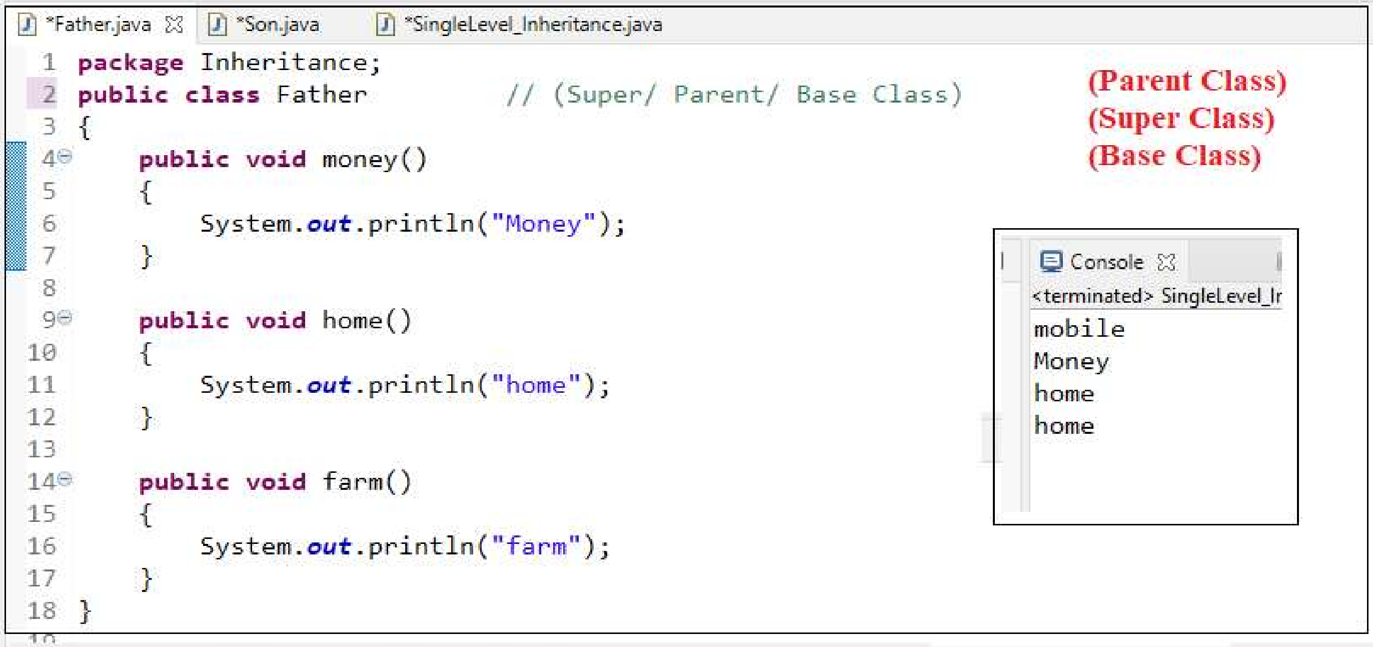
(Class:2) (Child/Sub Class)

```
package Inheritance;                  //child / sub class
public class son extends father       (extends used to acquire properties of father into son)
{
      public void mobile()
      {
               System.out.println("mobile");
      }                               (After using extend keyword following properties are
//    public void money()             present in class but not visible)
//    {
//             System.out.println("money");
//    }
//    public void car()
//    {
//             System.out.println("car");
//    }
//    public void home()
//    {
//             System.out.println("home");
//    }
}
```

(Class:3) (Only for Execution)

```
package Inheritance;
public class singleLevelInheritance {
public static void main(String[] args) {
      son s = new son();              (You can Create object of Any Class, depends upon use)
      s.mobile();
      s.money();
      s.car();
      s.home();
}
}
```

Single-level Inheritance



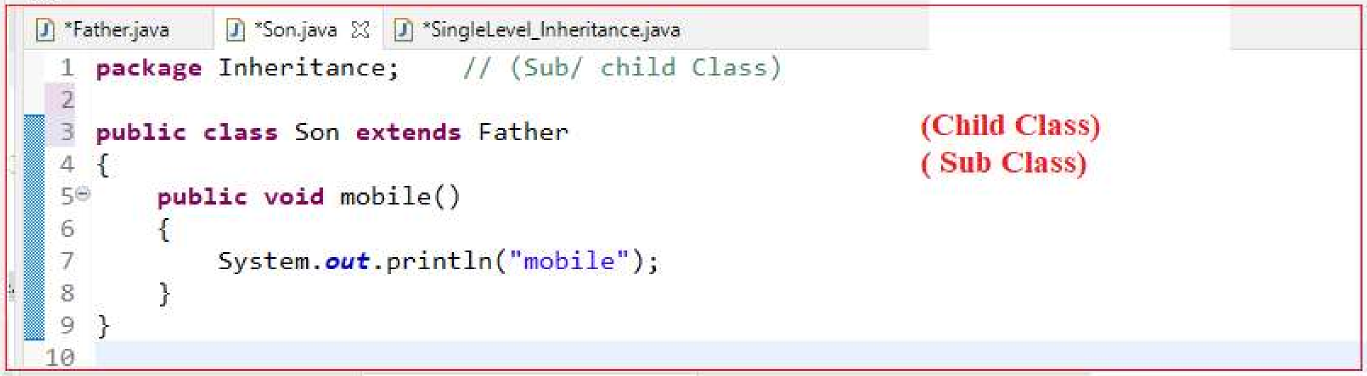
*Father.java | *Son.java | *SingleLevel_Inheritance.java

```java
package Inheritance;
public class Father          // (Super/ Parent/ Base Class)
{
    public void money()
    {
        System.out.println("Money");
    }

    public void home()
    {
        System.out.println("home");
    }

    public void farm()
    {
        System.out.println("farm");
    }
}
```

(Parent Class)
(Super Class)
(Base Class)

Console

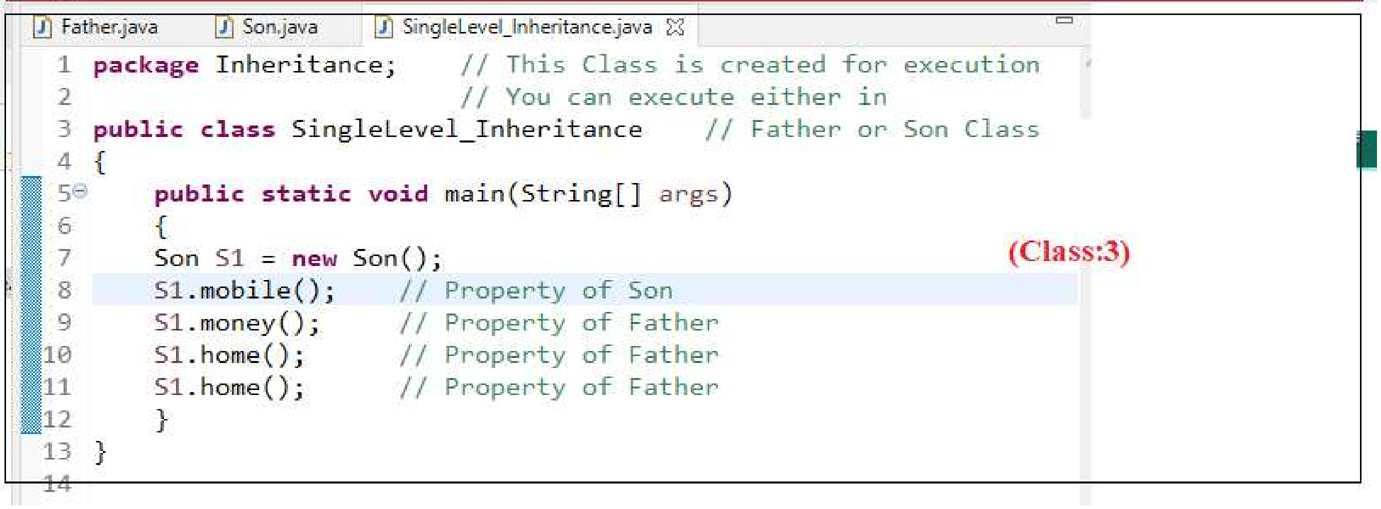
```
<terminated> SingleLevel_Ir
mobile
Money
home
home
```



*Father.java | *Son.java | *SingleLevel_Inheritance.java

```java
package Inheritance;    // (Sub/ child Class)

public class Son extends Father
{
    public void mobile()
    {
        System.out.println("mobile");
    }
}
```

(Child Class)
(Sub Class)



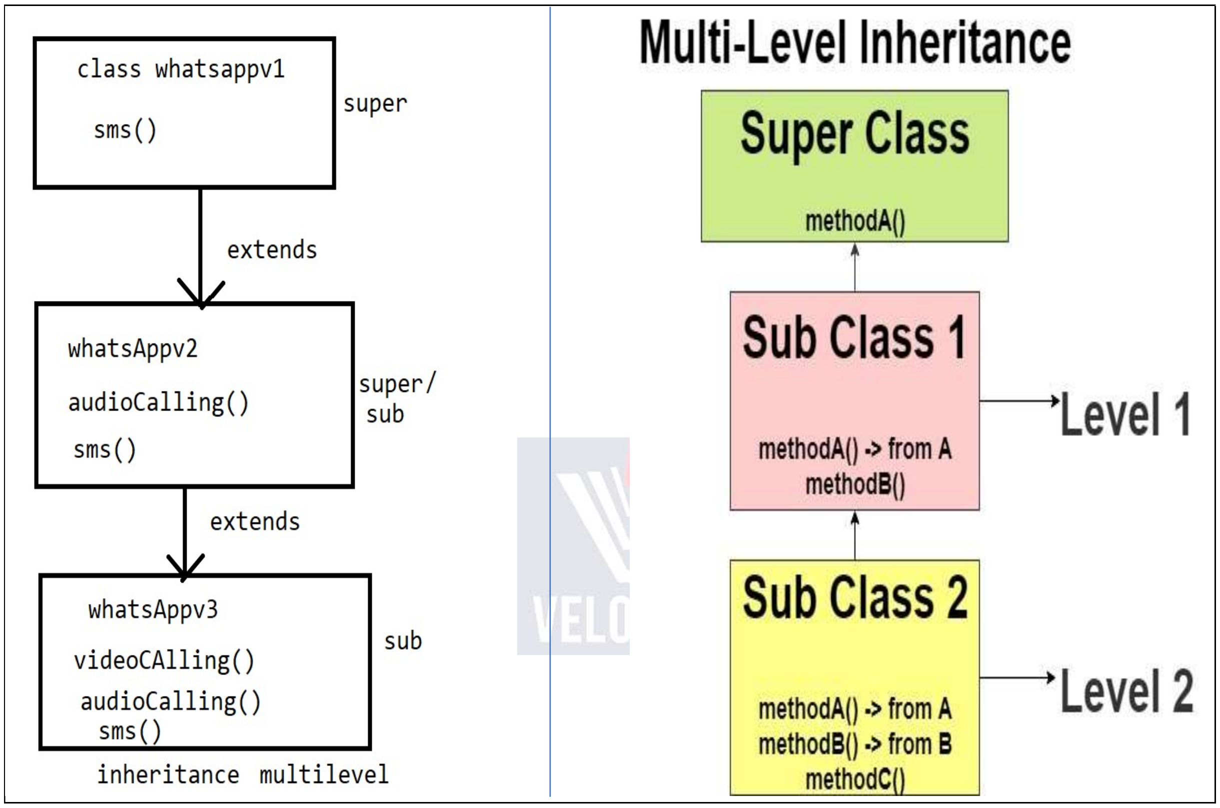
Father.java | Son.java | SingleLevel_Inheritance.java

```java
package Inheritance;     // This Class is created for execution
                         // You can execute either in
public class SingleLevel_Inheritance    // Father or Son Class
{
    public static void main(String[] args)
    {
    Son S1 = new Son();
    S1.mobile();     // Property of Son
    S1.money();      // Property of Father
    S1.home();       // Property of Father
    S1.home();       // Property of Father
    }
}
```

(Class:3)

2. Multi-level Inheritance:

- **Multilevel Inheritance takes place between 3 or more than 3 classes.**
- **In Multilevel Inheritance 1 sub class acquires properties of another super class & that class acquires properties of its another super class & phenomenon continuous.**
- **In the Multilevel inheritance, a derived class will inherit a base class and as well as the derived class also act as the base class to other class**



Example: Multilevel Inheritance **(Super Class)**

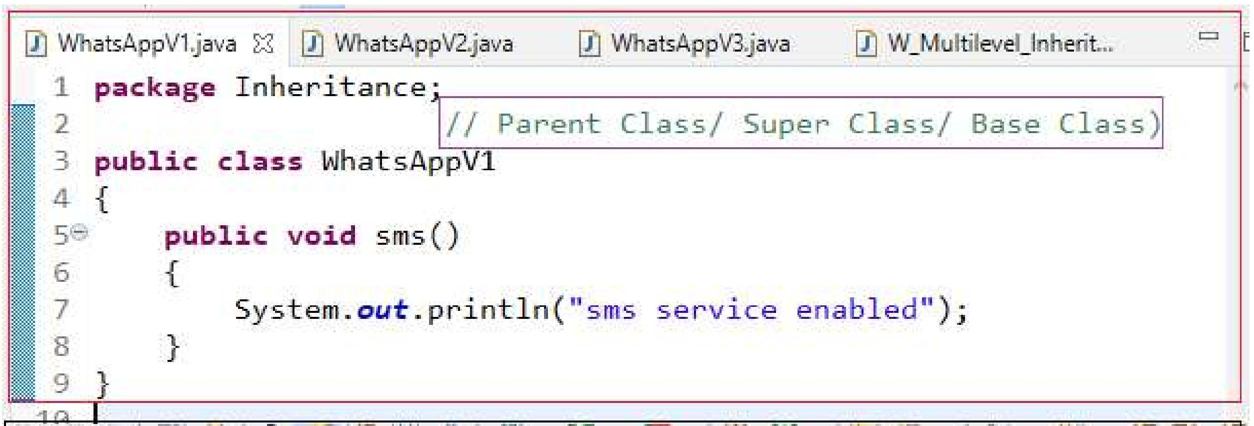
```
package Inheritance;
public class WhatsAppV1
{
	public void sms()
	{
		System.out.println("sms");
	}
}
```

```
package Inheritance;                                    (Super or sub class)

public class WhatsAppV2 extends WhatsAppV1              (Subclass extends Superclass)
{
        public void audioCalling()
        {
                System.out.println("audio Calling");
        }
//      public void sms()
//      {
//              System.out.println("sms");
//      }
}
```

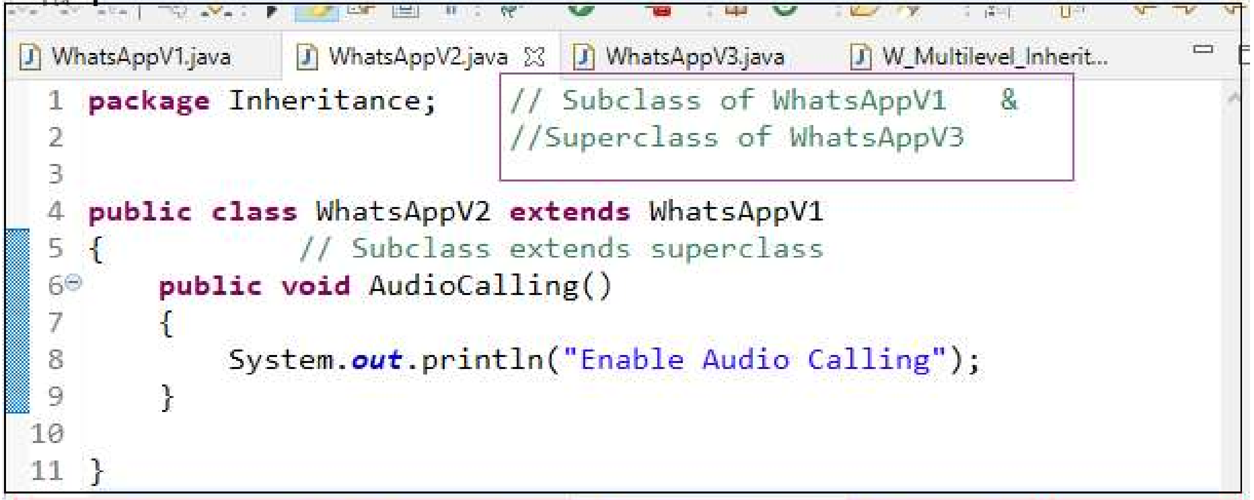
```
package Inheritance;                                    (Subclass)

public class WhatsAppV3 extends WhatsAppV2
{
        public void videoCalling()
        {
                System.out.println("video Calling");
        }
//      public void audioCalling()
//      {
//              System.out.println("audio Calling");
//      }
//      public void sms()
//      {
//              System.out.println("sms");
//      }
}
```

```
package Inheritance;

public class Multilevel_Inheritance
{
        public static void main(String[] args)
        {
                WhatsAppV3 v3 = new WhatsAppV3();
                v3.sms();
                v3.AudioCalling();
                v3.VideoCalling();
        }
}
```

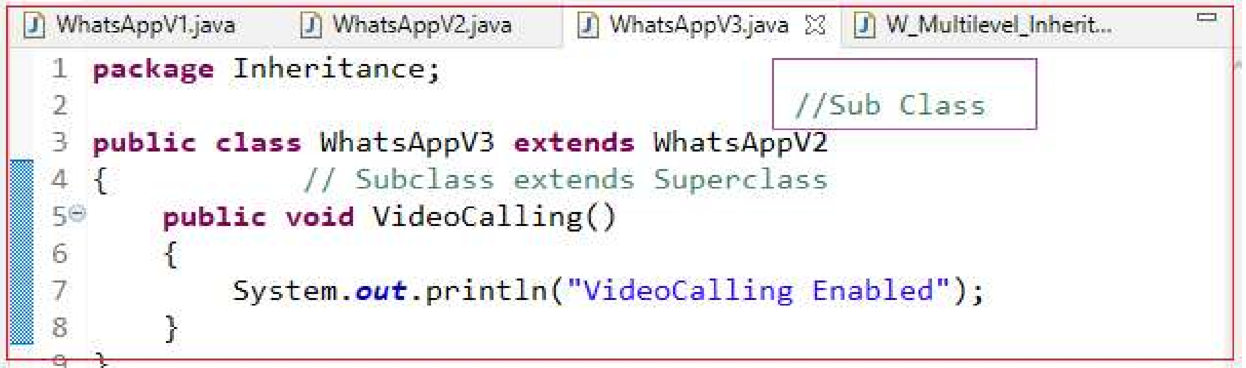
Multilevel Inheritance



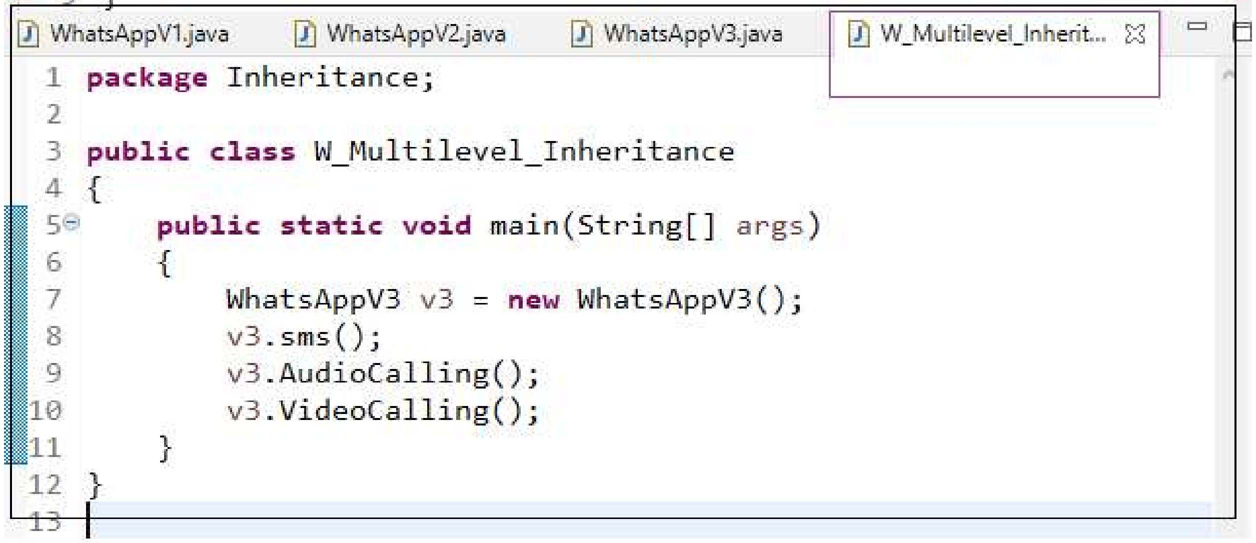
```java
package Inheritance;
                                     // Parent Class/ Super Class/ Base Class)
public class WhatsAppV1
{
    public void sms()
    {
        System.out.println("sms service enabled");
    }
}
```



```java
package Inheritance;         // Subclass of WhatsAppV1   &
                             //Superclass of WhatsAppV3

public class WhatsAppV2 extends WhatsAppV1
{            // Subclass extends superclass
    public void AudioCalling()
    {
        System.out.println("Enable Audio Calling");
    }

}
```



```java
package Inheritance;
                                         //Sub Class
public class WhatsAppV3 extends WhatsAppV2
{            // Subclass extends Superclass
    public void VideoCalling()
    {
        System.out.println("VideoCalling Enabled");
    }
}
```

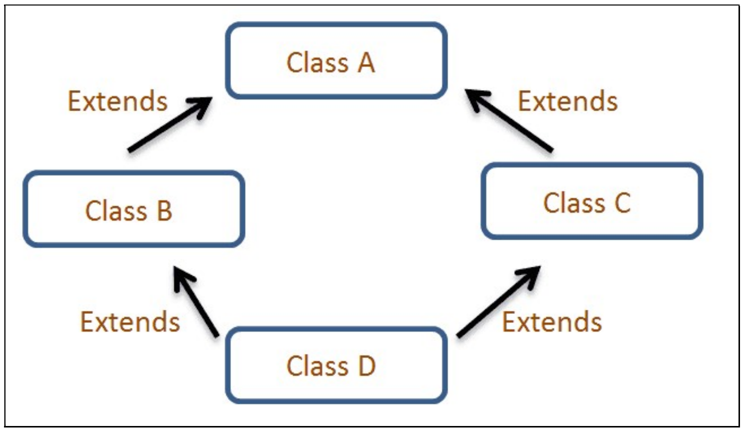


```java
package Inheritance;

public class W_Multilevel_Inheritance
{
    public static void main(String[] args)
    {
        WhatsAppV3 v3 = new WhatsAppV3();
        v3.sms();
        v3.AudioCalling();
        v3.VideoCalling();
    }
}
```

Multiple Inheritance:

- **If one sub class acquiring properties of two super class at the same time then it is referred as Multiple Inheritance**
- **Multiple Inheritance can be achieved by using interface.**
- **Java doesn't support Multiple inheritance using class because of "Diamond Ambiguity" problem.**

NOTE: Object class is the super most class in java.



---

3. Hierarchical Inheritance:

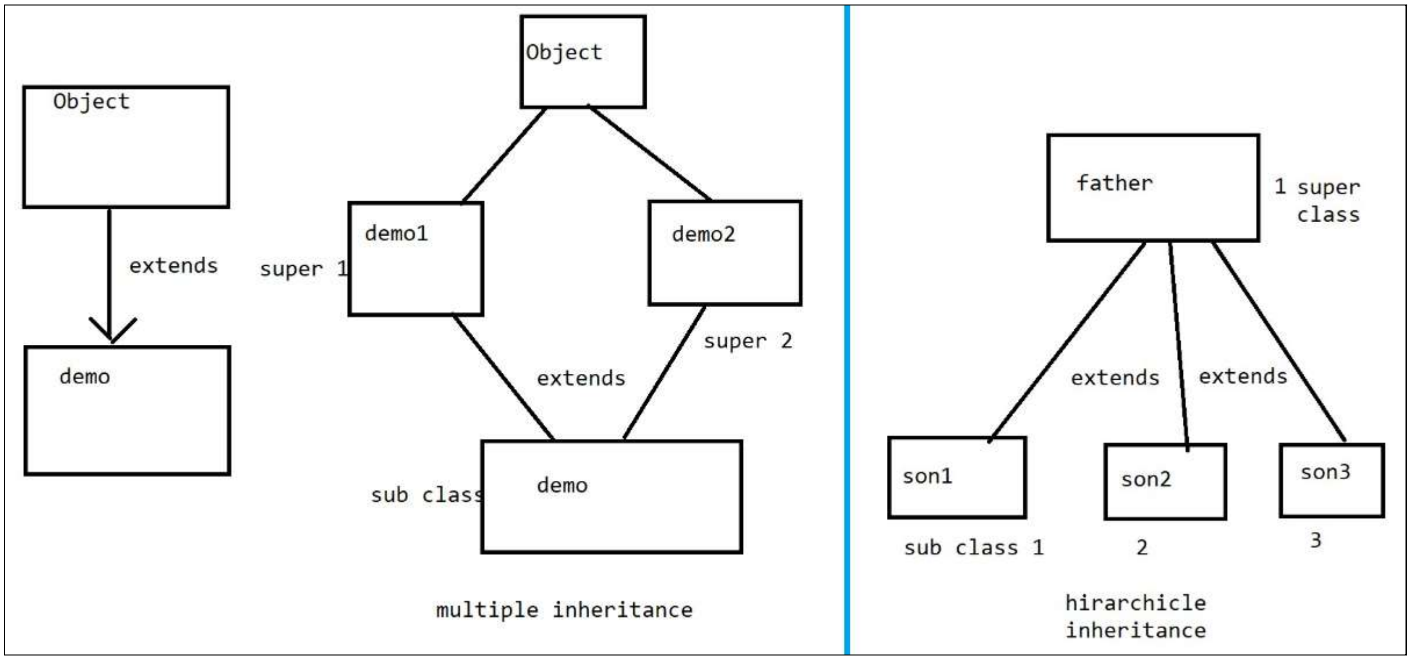
When multiple sub classes can acquire properties of 1 super class is known as hierarchical inheritance.

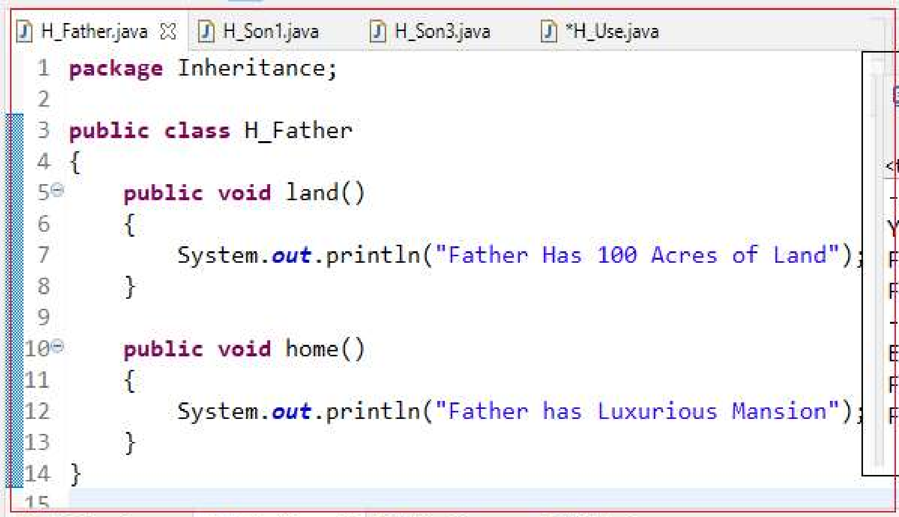
```
package Inheritance;                    (//parent / base / super)
public class father {
            public void money()
            {
                        System.out.println("money");
            }

            public void car()
            {
                        System.out.println("car");
            }
            public void home()
            {
                        System.out.println("home");
            }
}
```

```
package Inheritance;                    (//sub class1)
public class son1 extends father
{
	public void mobile()
	{
		System.out.println("mobile");
	}
//	public void money()
//	{
//		System.out.println("money");
//	}
//
//	public void car()
//	{
//		System.out.println("car");
//	}
//
//	public void home()
//	{
//		System.out.println("home");
//	}
}
```

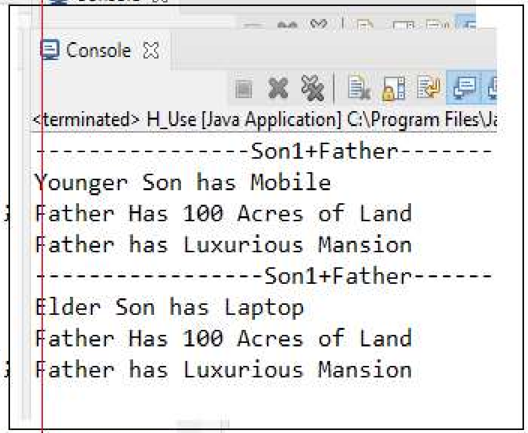
```
package Inheritance;                    (//sub class3)

public class son3 extends father
{
	public void laptop()
	{
		System.out.println("laptop");
	}

//	public void money()
//	{
//		System.out.println("money");
//	}
//
//	public void car()
//	{
//		System.out.println("car");
//	}
//
//	public void home()
//	{
//		System.out.println("home");
//	}
}
```

```
package Inheritance;

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

        System.out.println("-----properties of son1-----");

        son1 s1=new son1();
        s1.mobile();
        s1.car();
        s1.money();
        s1.home();

        System.out.println("-----properties of son2-----");

        son2 s2=new son2();
        s2.bike();
        s2.car();
        s2.money();
        s2.home();

        System.out.println("-----properties of son3-----");
        son3 s3=new son3();
        s3.laptop();
        s3.car();
        s3.money();
        s3.home();
}
}
```

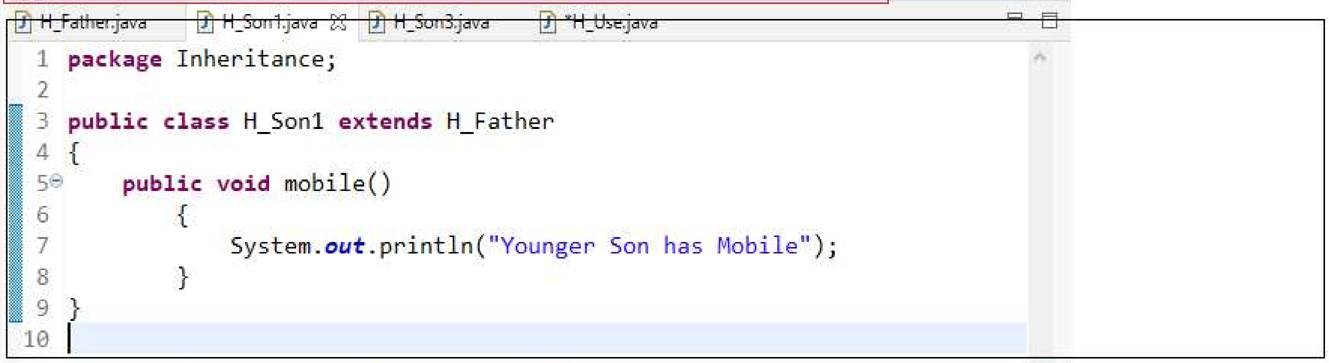




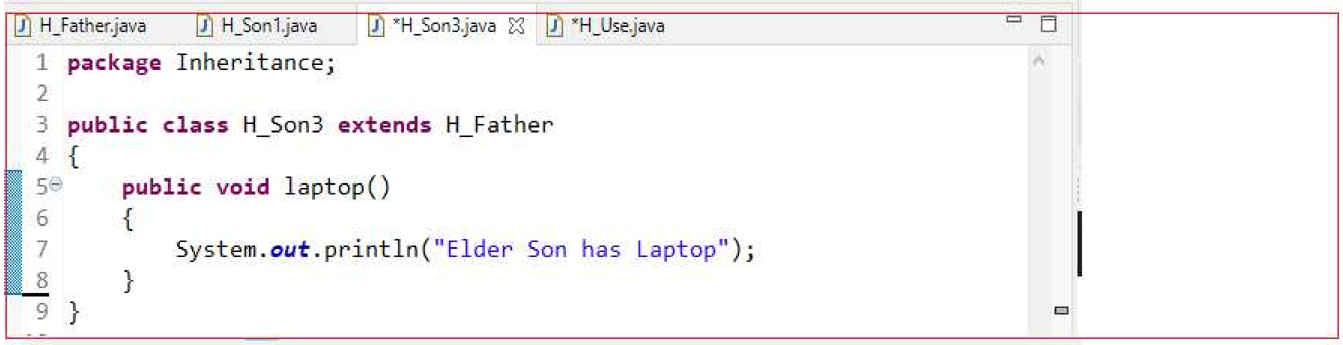
```java
package Inheritance;

public class H_Father
{
    public void land()
    {
        System.out.println("Father Has 100 Acres of Land");
    }

    public void home()
    {
        System.out.println("Father has Luxurious Mansion");
    }
}
```



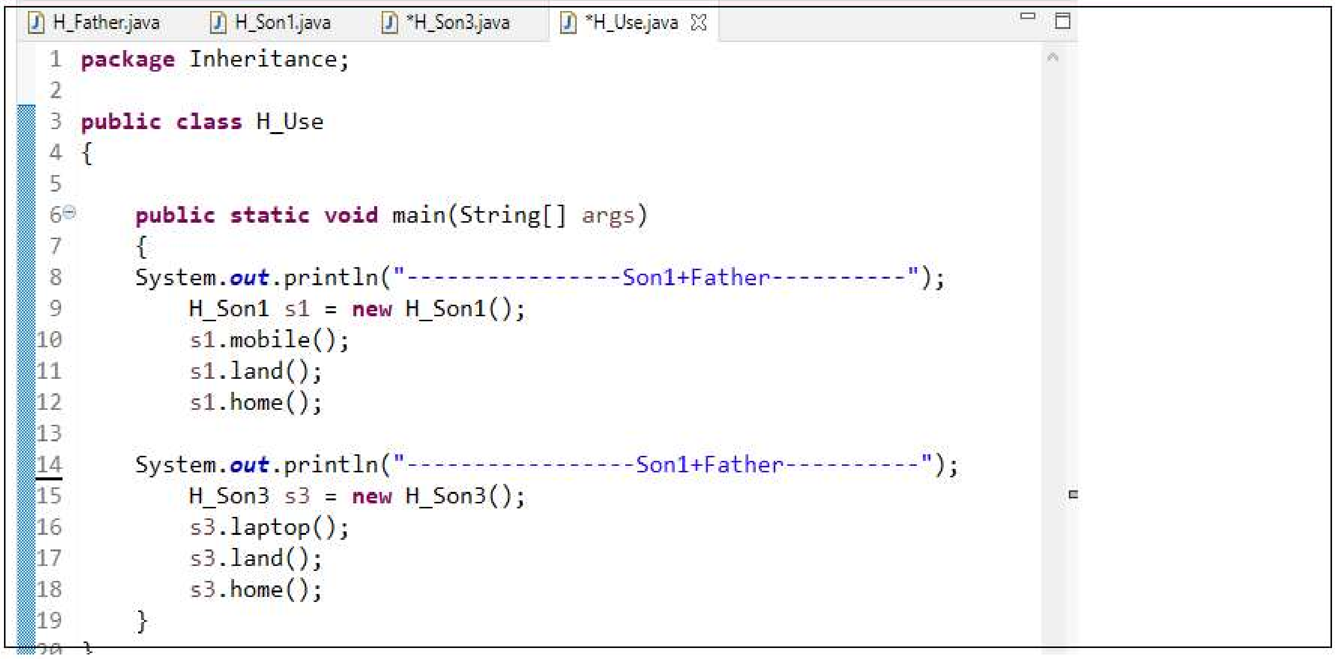
```
<terminated> H_Use [Java Application] C:\Program Files\Ja
---------------Son1+Father-------
Younger Son has Mobile
Father Has 100 Acres of Land
Father has Luxurious Mansion
----------------Son1+Father------
Elder Son has Laptop
Father Has 100 Acres of Land
Father has Luxurious Mansion
```



```java
package Inheritance;

public class H_Son1 extends H_Father
{
    public void mobile()
        {
            System.out.println("Younger Son has Mobile");
        }
}
```



```java
package Inheritance;

public class H_Son3 extends H_Father
{
    public void laptop()
    {
        System.out.println("Elder Son has Laptop");
    }
}
```



```java
package Inheritance;

public class H_Use
{

    public static void main(String[] args)
    {
    System.out.println("---------------Son1+Father---------");
        H_Son1 s1 = new H_Son1();
        s1.mobile();
        s1.land();
        s1.home();

    System.out.println("----------------Son1+Father---------");
        H_Son3 s3 = new H_Son3();
        s3.laptop();
        s3.land();
        s3.home();
    }
}
```

This Keyword

This keyword is used to access global variable from same/current class.

Super Keyword

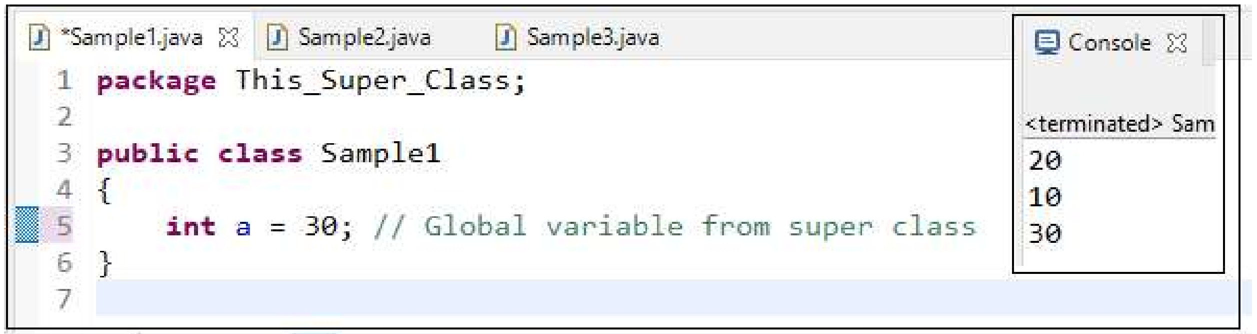
Super Keyword is used to access global variable from supper/different class.

```
package This_Super_Keyword;

public class sample1 extends sample
{
	//int a=30;   // global variable from super class

	int a=10;   //global variable from same/current class

	public void m1()
	{
	int a=20;  //local varaible
	System.out.println(a);  //20
	System.out.println(this.a);  //10   //call global variable from same/current class
	System.out.println(super.a);   //30  //call global variable from super class
	}
}
```

```
package This_Super_Keyword;

public class sample2 {
	public static void main(String[] args) {

		sample1 s1=new sample1();
		s1.m1();

	}
}
```

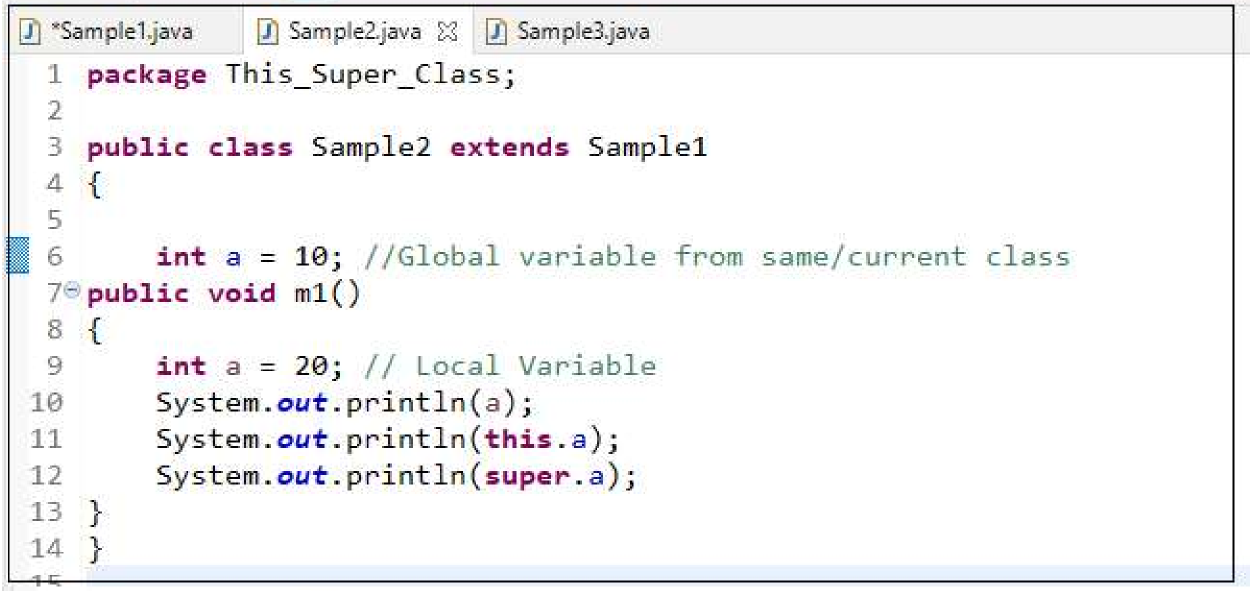
```
package This_Super_Keyword;

public class sample
{
	int a=30;
}
```

Use of This and Super Keyword

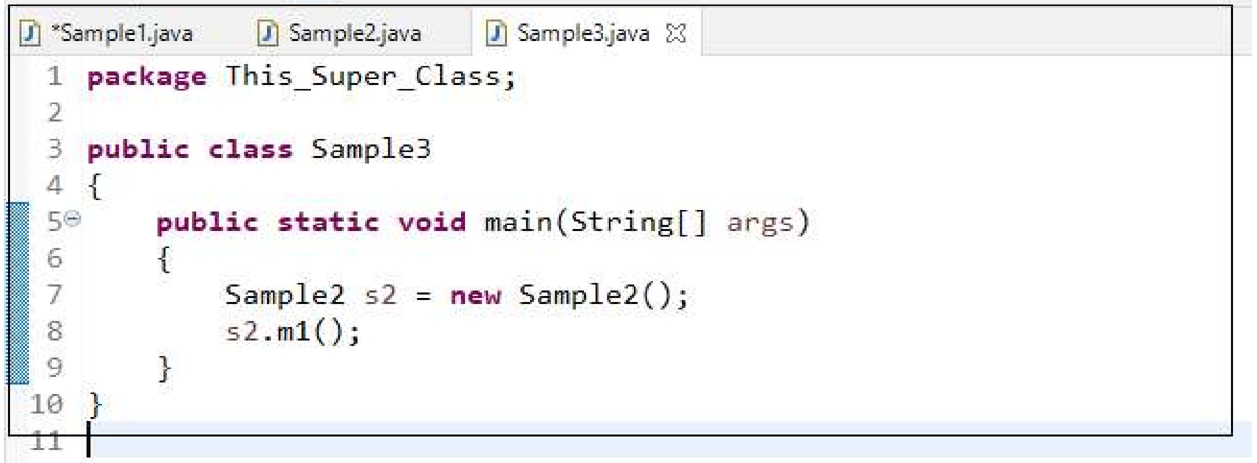


```java
package This_Super_Class;

public class Sample1
{
    int a = 30; // Global variable from super class
}
```

```
20
10
30
```



```java
package This_Super_Class;

public class Sample2 extends Sample1
{

    int a = 10; //Global variable from same/current class
public void m1()
{
    int a = 20; // Local Variable
    System.out.println(a);
    System.out.println(this.a);
    System.out.println(super.a);
}
}
```



```java
package This_Super_Class;

public class Sample3
{
    public static void main(String[] args)
    {
        Sample2 s2 = new Sample2();
        s2.m1();
    }
}
```

Access specifiers

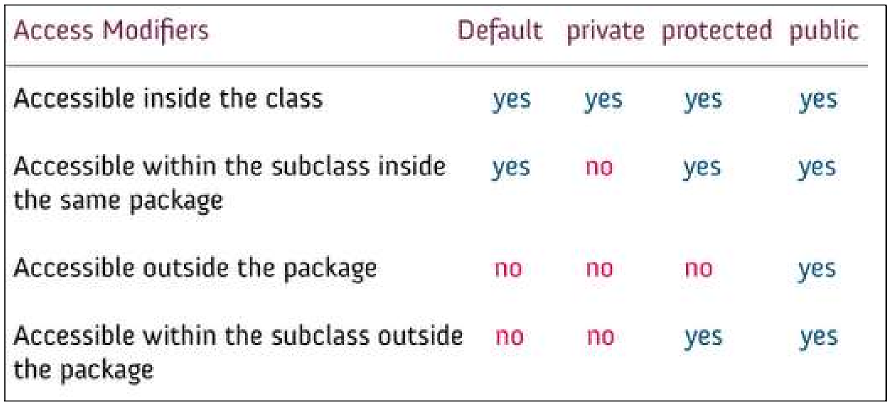
Access specifiers are used to represent scope of members of class.
In java Access specifiers are classified into 4 types
1. private
2. default
3. protected
4. public

1. **private: (within only class)**
    - **If you declare any member of class as private then scope of that member remains only within the class.**
    - **It can't be access from other classes.**
2. **default: (within package)**
    - **If you declare any member of class as default then scope of that member remains only within the package**
    - **It can't be access from other packages.**
    - **There is no keyword to represent default access specifier.**

3. **protected: (Within package/ In other package but inheritance mandatory)**
    - **If you declare any member of class as protected then scope of that member remains only within the package**
    - **That class which is present outside the package can access it by one condition ie. inheritance operation**

4. **public: (within project)**

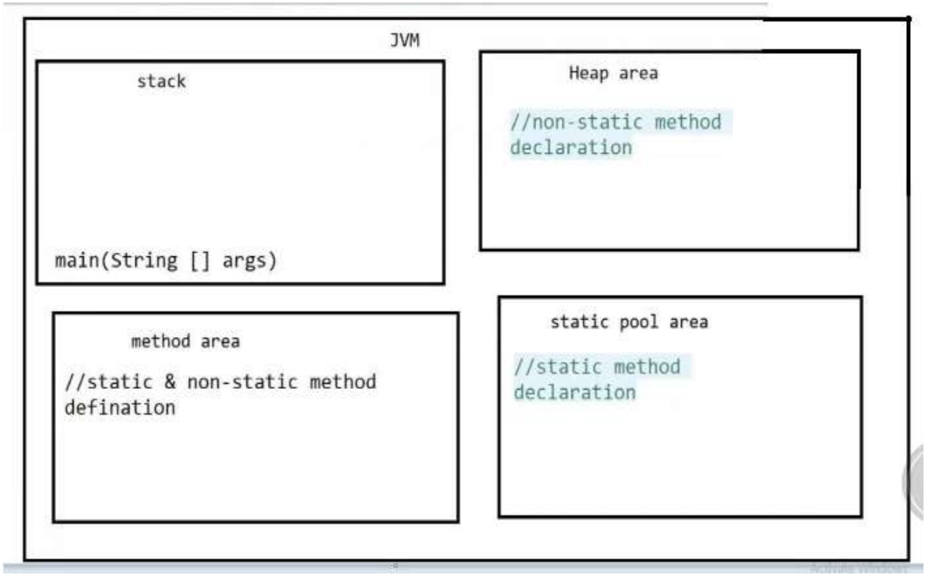
If you declare any member of class as public then scope of that member remains through the project.



Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes

Diff types of JVM memories:

1. Heap area--> non-static method declaration.
2. Static pool area--> static method declaration.
3. method area --> static & non-static method definition.
4. stack --> main()--> method execution flow.


JVM
stack
main(String [] args)
Heap area
//non-static method declaration
method area
//static & non-static method defination
static pool area
//static method declaration

Polymorphism:

- It is one of the OOPs principle where one object showing different behavior at different stages of life cycle.
- Polymorphism is a Latin word where poly stand for many & morphism stands for forms.
- In java Polymorphism is classified into 2 types:
  1. Compiletime Polymorphism
  2. Runtime Polymorphism

1. Compiletime Polymorphism:

- In Compiletime Polymorphism method declaration is going to get binded to its definition at compilation time, based on argument/input/parameter is known as compiletime Polymorphism.
- As binding takes during compilation time only, so it is also known as early binding.
- Once binding is done, rebinding can't be done, so it is called static binding.
- Method overloading is an example of compiletime Polymorphism

Method overloading:
Declaring multiple method with same method name but with different argument/parameter/inputs in a same class is called method overloading.

2. Runtime Polymorphism:

- **In Runtime Polymorphism method declaration is going to get binded to its definition at Runtime/execution time, based on object creation is known as runtime Polymorphism.**
- **As binding takes during Runtime/execution time, so it is also known as late binding.**
- **Once binding is done, rebinding can be done, so it is called dynamic binding.**
- **Method overriding is an example of Runtime Polymorphism.**

Method overriding:

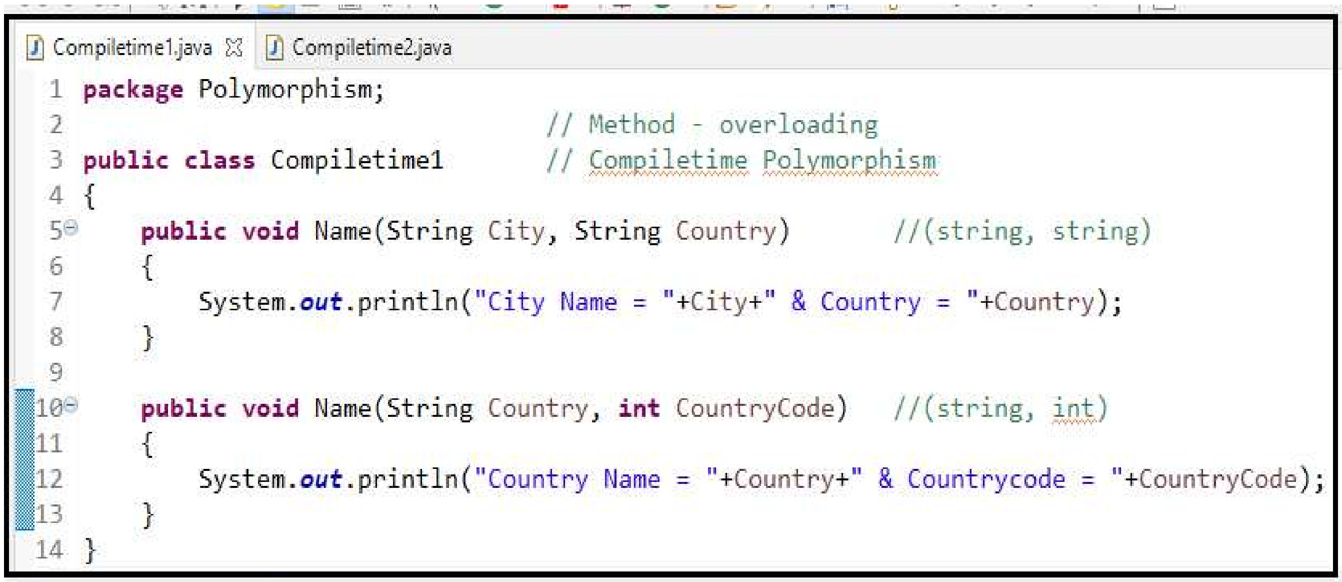
Acquiring super class method into sub class with the help of extends keyword & changing implementation/definition according to subclass specification is called method overriding

---

```
package PolyMorphism;              (//Method Overloading) (Compiletime Polymorphism)
public class demo1
{
        public void addition(int a, int b)   // 2 int parameter
        {
                int sum=a+b;
                System.out.println(sum);
        }
        public void addition(int a, int b, int c)  //3 int parameter
        {
                int sum=a+b+c;
                System.out.println(sum);
        }
}
```

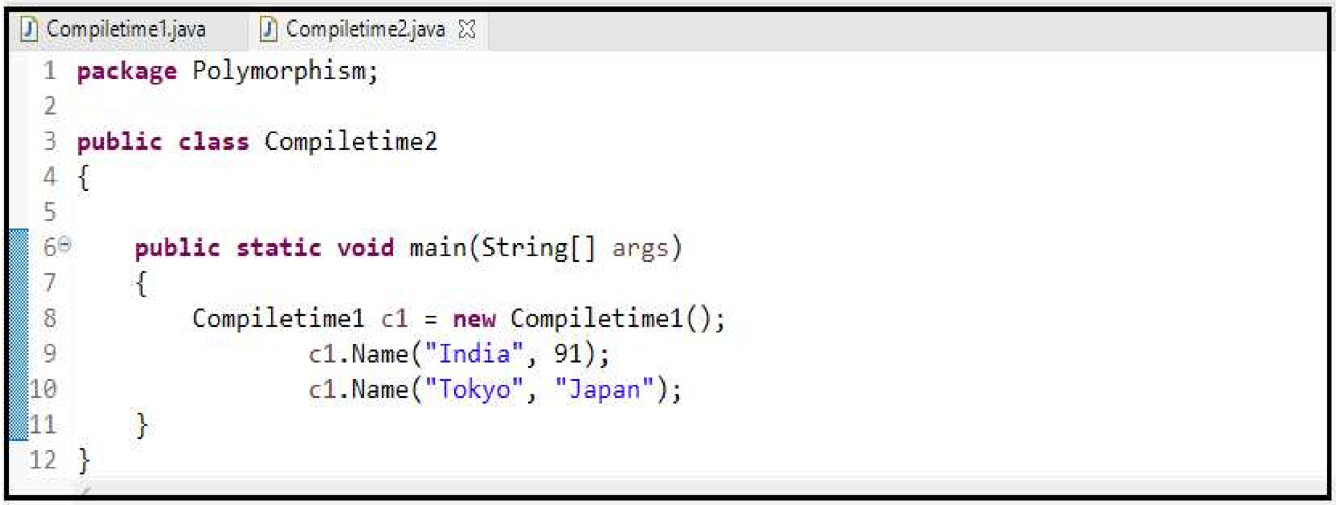
```
package PolyMorphism;
public class TestDemo
{
public static void main(String[] args)
{
        demo1 d1=new demo1();
        d1.addition(10,30);            //40
        d1.addition(5,6,7);            //18
}
}
```

Method overloading: // Compiletime Polymorphism



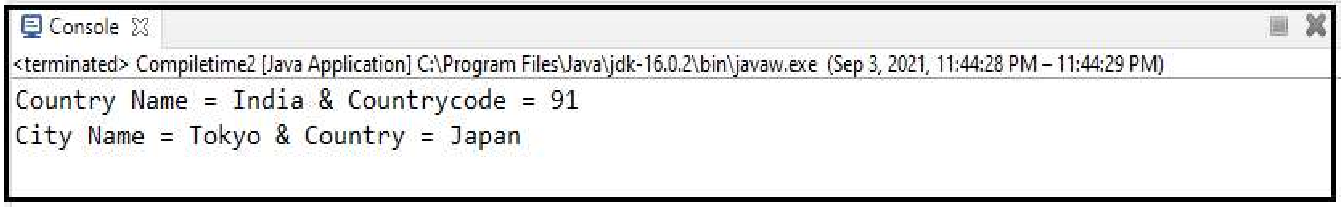
Compiletime1.java | Compiletime2.java

```java
package Polymorphism;
                                    // Method - overloading
public class Compiletime1           // Compiletime Polymorphism
{
    public void Name(String City, String Country)       //(string, string)
    {
        System.out.println("City Name = "+City+" & Country = "+Country);
    }

    public void Name(String Country, int CountryCode)   //(string, int)
    {
        System.out.println("Country Name = "+Country+" & Countrycode = "+CountryCode);
    }
}
```



Compiletime1.java | Compiletime2.java

```java
package Polymorphism;

public class Compiletime2
{

    public static void main(String[] args)
    {
        Compiletime1 c1 = new Compiletime1();
                c1.Name("India", 91);
                c1.Name("Tokyo", "Japan");
    }
}
```



Console

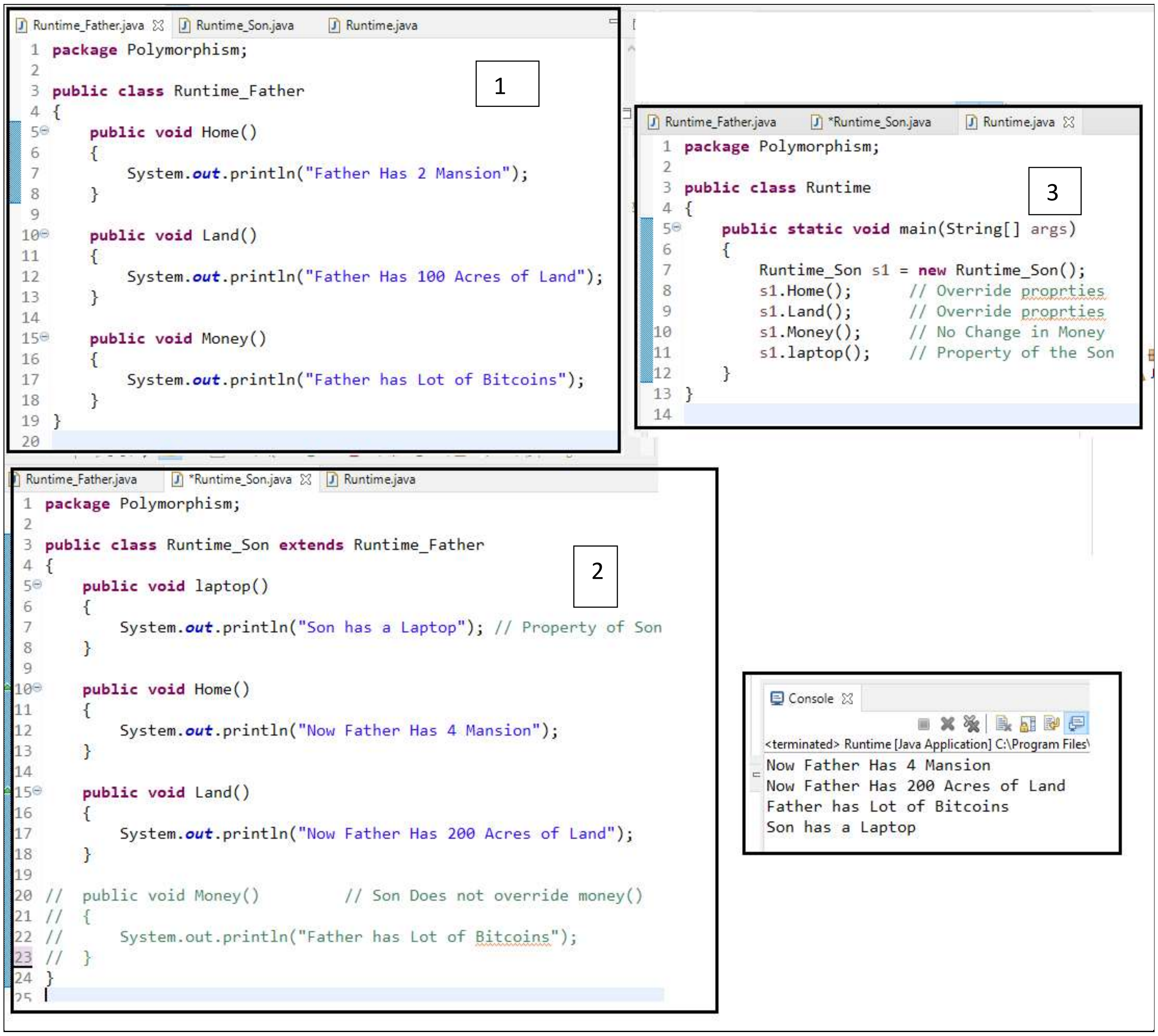
```
<terminated> Compiletime2 [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (Sep 3, 2021, 11:44:28 PM – 11:44:29 PM)
Country Name = India & Countrycode = 91
City Name = Tokyo & Country = Japan
```

```
package PolyMorphism;      (//Method Over-riding) (Runtime Polymorphism)
public class father             //parent / base / super
{
          public void money()
          {
                    System.out.println("money: 1L");
          }

          public void car()
          {
                    System.out.println("car: honda city");
          }

          public void home()
          {
                    System.out.println("home: 2BHK");
          }
}
```

```
package PolyMorphism;                              //child / sub class
public class son extends father
{
          public void money()                                //override
          {
                    System.out.println("money: 2L");
          }

          public void car()                                  //override
          {
                    System.out.println("car: kia seltos");
          }
//        public void home()                                 // No overriding
//        {
//                  System.out.println("home: 2BHK");
//        }

}
```

```
package PolyMorphism;
public class TestOverriding
{
          public static void main(String[] args)
          {
                    son s=new son();
                    s.money();
                    s.car();
                    s.home();
          }
}
```

(//Method Over-riding) (Runtime Polymorphism)



```
package Polymorphism;

public class Runtime_Father
{
    public void Home()
    {
        System.out.println("Father Has 2 Mansion");
    }

    public void Land()
    {
        System.out.println("Father Has 100 Acres of Land");
    }

    public void Money()
    {
        System.out.println("Father has Lot of Bitcoins");
    }
}
```

```
package Polymorphism;

public class Runtime_Son extends Runtime_Father
{
    public void laptop()
    {
        System.out.println("Son has a Laptop"); // Property of Son
    }

    public void Home()
    {
        System.out.println("Now Father Has 4 Mansion");
    }

    public void Land()
    {
        System.out.println("Now Father Has 200 Acres of Land");
    }

//  public void Money()          // Son Does not override money()
//  {
//      System.out.println("Father has Lot of Bitcoins");
//  }
}
```

```
package Polymorphism;

public class Runtime
{
    public static void main(String[] args)
    {
        Runtime_Son s1 = new Runtime_Son();
        s1.Home();      // Override proprties
        s1.Land();      // Override proprties
        s1.Money();     // No Change in Money
        s1.laptop();    // Property of the Son
    }
}
```

```
<terminated> Runtime [Java Application] C:\Program Files\
Now Father Has 4 Mansion
Now Father Has 200 Acres of Land
Father has Lot of Bitcoins
Son has a Laptop
```

Abstract Class:

- **A class declared with "abstract" keyword is called abstract class.**
- **An Abstract class is nothing but an incomplete class where programmer can declare complete as well as incomplete methods in it. (It requires Min 1 Complete and 1 Incomplete Method)**
- **Programmer can declare incomplete methods as abstract method, by declaring keyword called "abstract" Infront of method.**
- **We can't create object of abstract class, to create object of abstract class we need to make use of concrete class.**

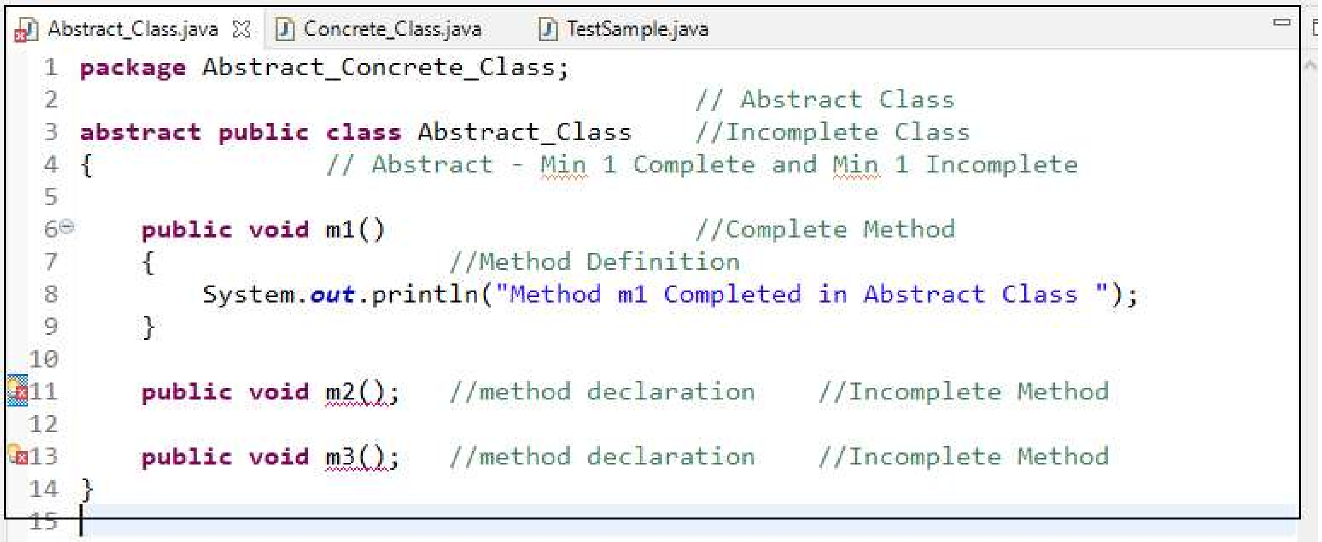
Concrete class:

A class which provides definitions for all the incomplete methods which are present in abstract class with the help of extends keywords is called concrete class.

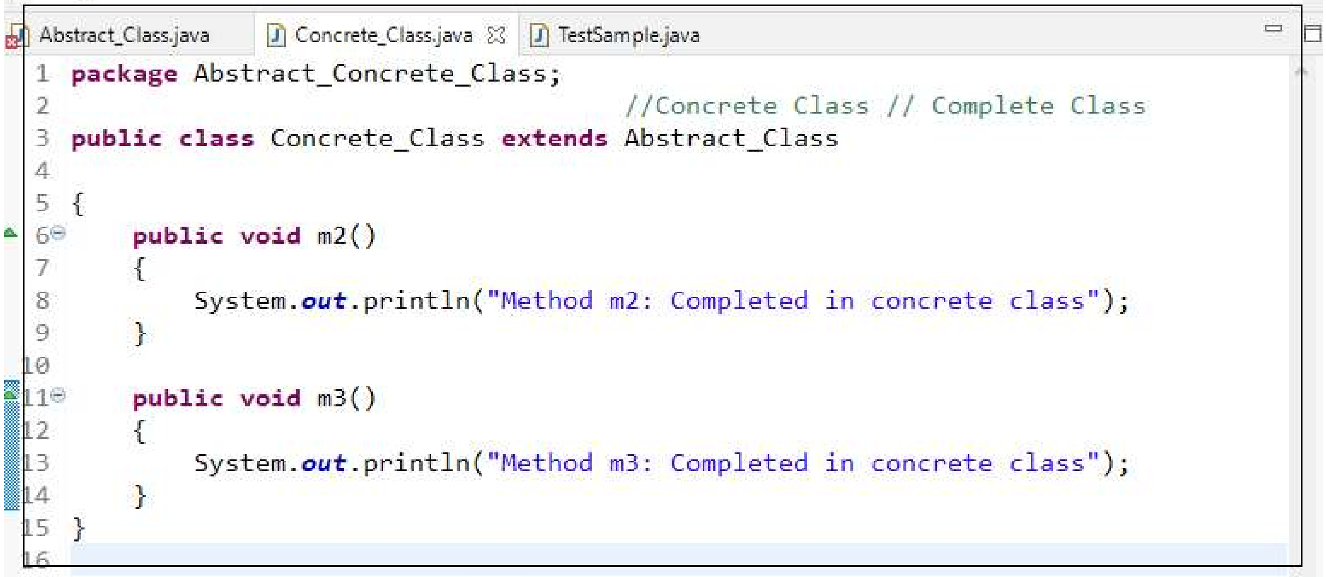
```
package Abstract_Concrete_Class;                    //incomplete class --> abstract class
abstract public class sample1
{
        //complete method
        public void m1()                                //method declaration
        {                                               //method definition
                System.out.println(" method m1: completed in abstract class");
        }

        //incomplete method
        abstract public void m2();                      //method declaration

        //incomplete method
        abstract public void m3();                      //method declaration

}
```

```
package Abstract_Concrete_Class;              //concrete class--> complete class
public class sample2 extends sample1
{
        public void m2()
        {
                System.out.println("method m2: completed in concrete class");
        }

        public void m3()
        {
                System.out.println("method m3: completed in concrete class");
        }

//      public void m1()                        //method declaration
//      {                                       //method definition
//              System.out.println(" method m1: completed in abstract class");
//      }
}
```

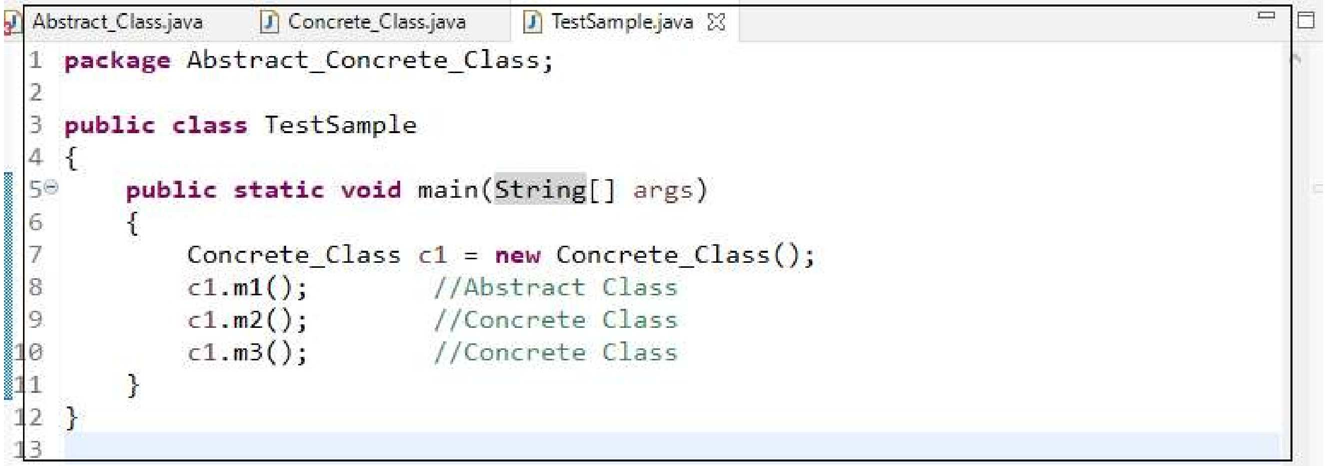
```
package Abstract_Concrete_Class;
public class TestSample
{
public static void main(String[] args)
        {
                sample2 s2=new sample2();
                s2.m1();
                s2.m2();
                s2.m3();
        }
}
```


Abstract_Class.java | Concrete_Class.java | TestSample.java

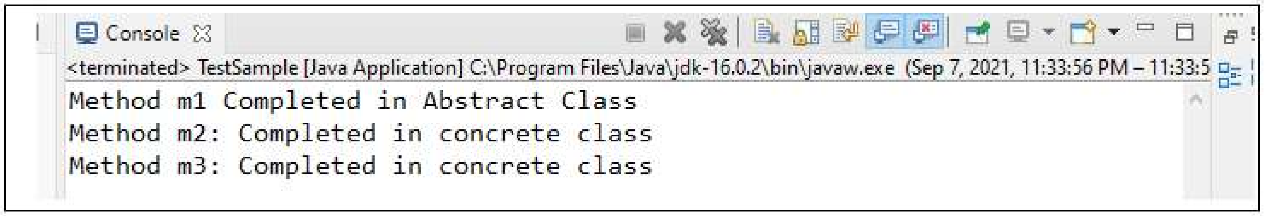
```java
package Abstract_Concrete_Class;
                                                // Abstract Class
abstract public class Abstract_Class        //Incomplete Class
{                   // Abstract - Min 1 Complete and Min 1 Incomplete

    public void m1()                            //Complete Method
    {                       //Method Definition
        System.out.println("Method m1 Completed in Abstract Class ");
    }

    public void m2();   //method declaration    //Incomplete Method

    public void m3();   //method declaration    //Incomplete Method
}
```


Abstract_Class.java | Concrete_Class.java | TestSample.java

```java
package Abstract_Concrete_Class;
                                    //Concrete Class // Complete Class
public class Concrete_Class extends Abstract_Class

{
    public void m2()
    {
        System.out.println("Method m2: Completed in concrete class");
    }

    public void m3()
    {
        System.out.println("Method m3: Completed in concrete class");
    }
}
```


Abstract_Class.java | Concrete_Class.java | TestSample.java

```java
package Abstract_Concrete_Class;

public class TestSample
{
    public static void main(String[] args)
    {
        Concrete_Class c1 = new Concrete_Class();
        c1.m1();        //Abstract Class
        c1.m2();        //Concrete Class
        c1.m3();        //Concrete Class
    }
}
```


Console

```
<terminated> TestSample [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (Sep 7, 2021, 11:33:56 PM – 11:33:5
Method m1 Completed in Abstract Class
Method m2: Completed in concrete class
Method m3: Completed in concrete class
```

Interface:

- It is one of the oops principle.
- It is pure 100% abstract in nature.
- Interface is use to declare only incomplete methods in it.

Features of Interface:

1. Methods declared inside Interface are by default public & abstract.

2. Data Members/variable declared inside Interface are by default static and final.

3. Constructor concept in not present inside Interface.

4. Object of Interface can't be created.

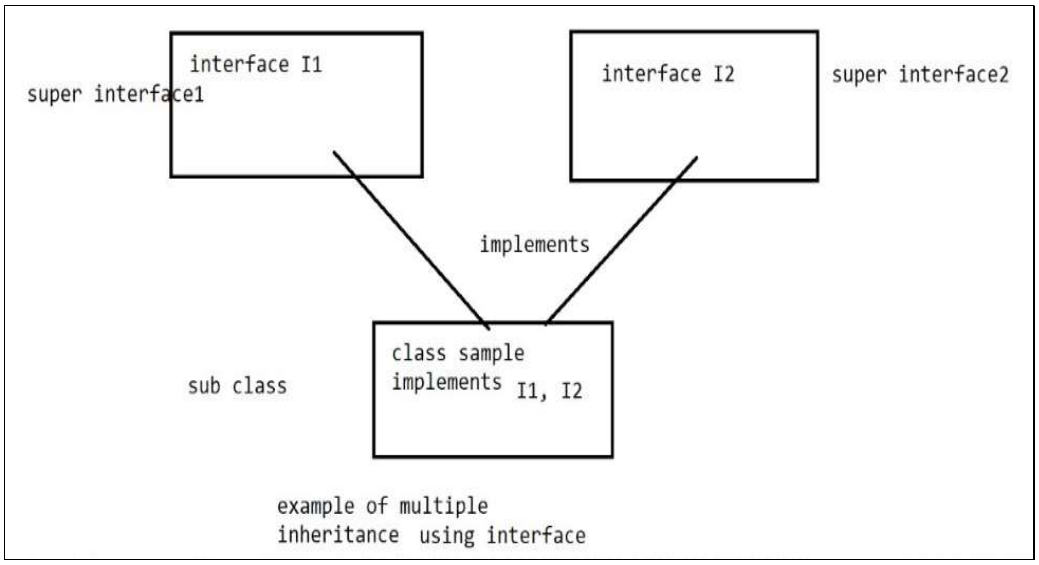
5. To create object of Interface programmer need to make use of Implementation class using implements keyword.

6. Interface support multiple inheritance.

Implementation class:

A class which provides definitions for all the incomplete methods which are present in interface with the help of "implements" keyword is called Implementation class.

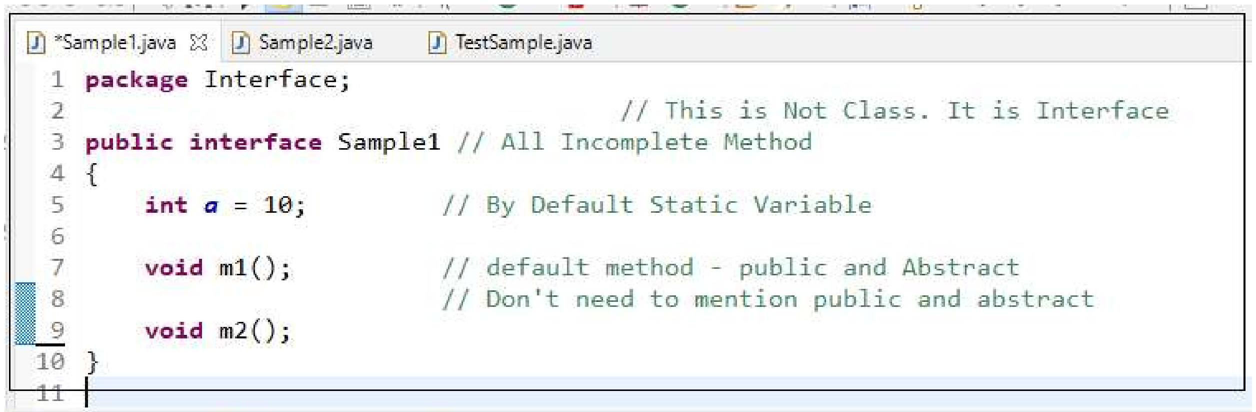
Multiple Inheritance Using Interface



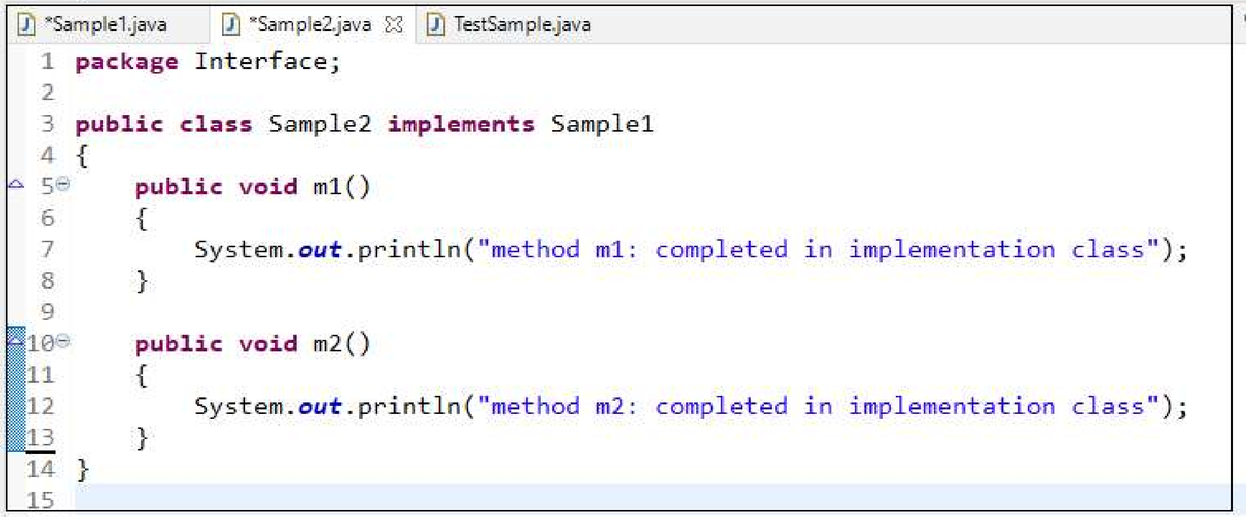
```
package Interface_ImplementationClass;                    //interfaceName--> demo
public interface demo
{
        // all incomplete methods
        int a=10;                      // static final int a=10;

        void m1();                     // public abstract void m1();

        void m2();                     // public abstract void m2();

}
```

```
package Interface_ImplementationClass;                    //demo1--> implementation class
public class demo1 implements demo
{
        public void m1()
        {
                System.out.println("method m1: completed in implementation class");
        }

        public void m2()
        {
                System.out.println("method m2: completed in implementation class");
        }
}
```

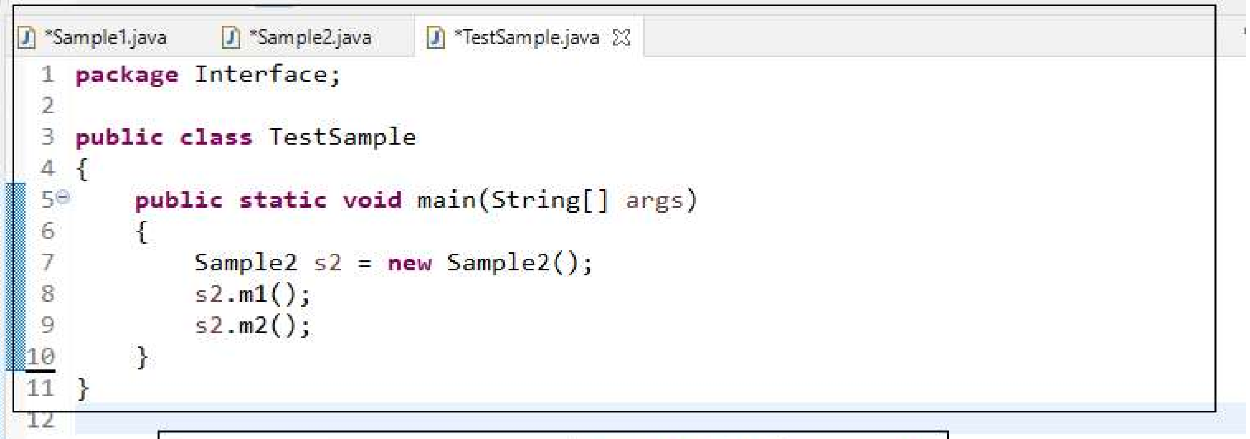
```
package Interface_ImplementationClass;
public class TestDemo1
{
public static void main(String[] args)       //example of interface & implementation class
        {
        demo1 d1=new demo1();
        d1.m1();
        d1.m2();
        }
}
```


*Sample1.java | Sample2.java | TestSample.java

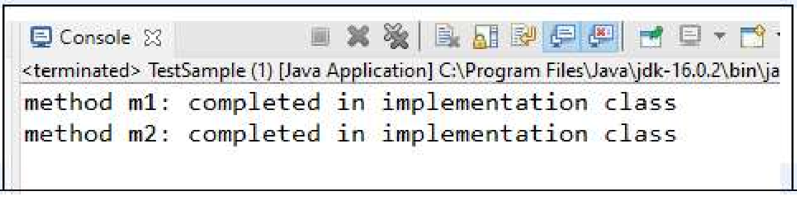
```java
package Interface;
                                  // This is Not Class. It is Interface
public interface Sample1 // All Incomplete Method
{
    int a = 10;          // By Default Static Variable

    void m1();           // default method - public and Abstract
                         // Don't need to mention public and abstract
    void m2();
}
```


*Sample1.java | *Sample2.java | TestSample.java

```java
package Interface;

public class Sample2 implements Sample1
{
    public void m1()
    {
        System.out.println("method m1: completed in implementation class");
    }

    public void m2()
    {
        System.out.println("method m2: completed in implementation class");
    }
}
```

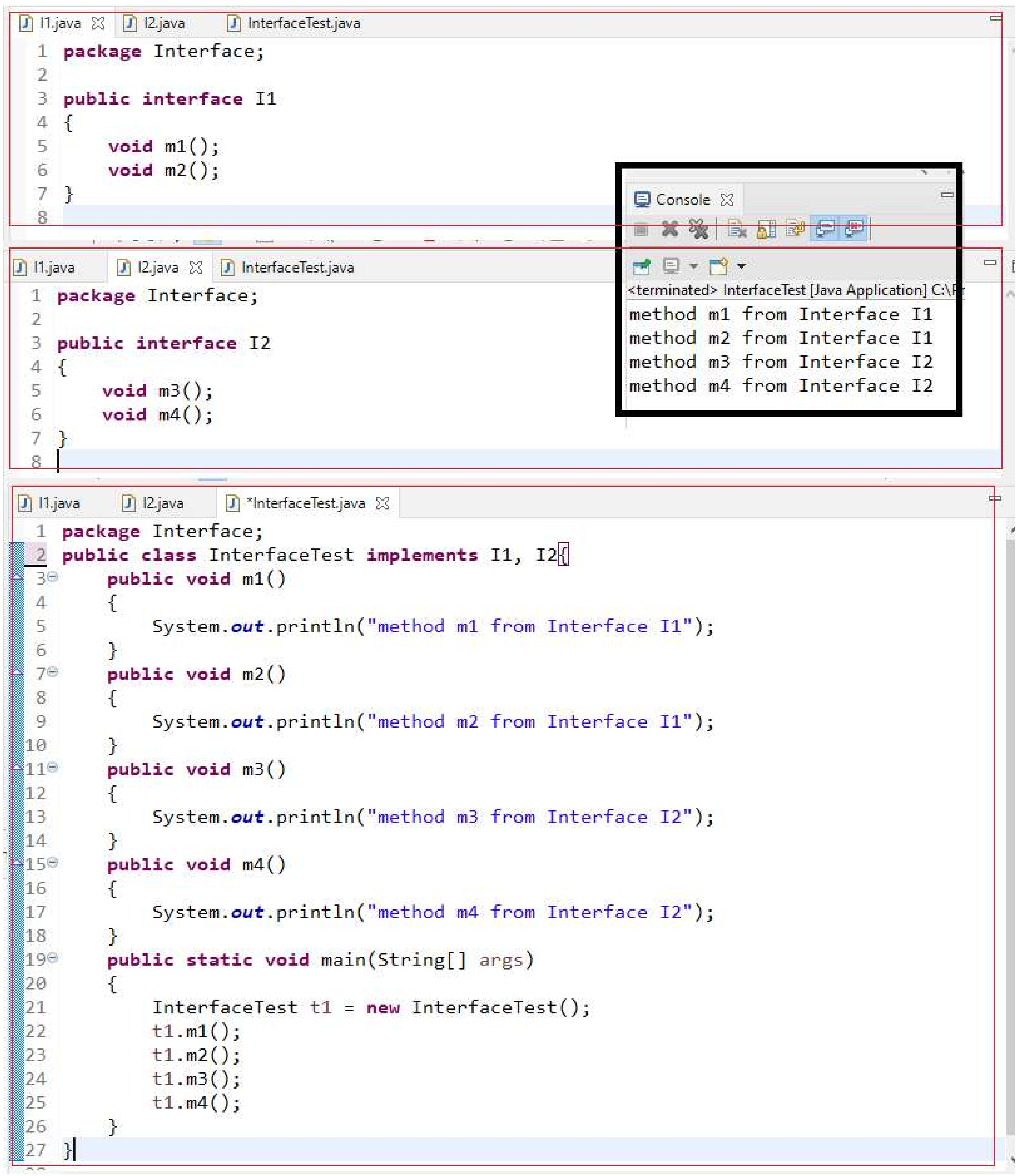

*Sample1.java | *Sample2.java | *TestSample.java

```java
package Interface;

public class TestSample
{
    public static void main(String[] args)
    {
        Sample2 s2 = new Sample2();
        s2.m1();
        s2.m2();
    }
}
```


Console

```
<terminated> TestSample (1) [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\ja
method m1: completed in implementation class
method m2: completed in implementation class
```

```
package Interface_ImplementationClass;                    //super interface1
public interface I1
{
        void m1();
        void m2();
}
                              -----------------------------------
package Interface_ImplementationClass;                    //super interface2
public interface I2
{
        void m3();
        void m4();
}
                              -----------------------------------
package Interface_ImplementationClass;  //sample1--> subclass--> implementation class
public class sample1 implements I1, I2
{
                                    //example of multiple inheritance using interface
        public void m1()
        {
                System.out.println("method m1 from Interface I1");
        }
        public void m2()
        {
                System.out.println("method m2 from Interface I1");
        }
        public void m3()
        {
                System.out.println("method m3 from Interface I2");
        }
        public void m4()
        {
                System.out.println("method m4 from Interface I2");
        }
}
                              ---------------------------------
package Interface_ImplementationClass;
public class TestSample1
{                       //example of multiple inheritance using interface
        public static void main(String[] args)
{
                sample1 s1=new sample1();
                s1.m1();
                s1.m2();
                s1.m3();
                s1.m4();
        }
}-----------------------------------------------------------------------------------------------
```



```java
package Interface;

public interface I1
{
    void m1();
    void m2();
}
```

```java
package Interface;

public interface I2
{
    void m3();
    void m4();
}
```

```
method m1 from Interface I1
method m2 from Interface I1
method m3 from Interface I2
method m4 from Interface I2
```

```java
package Interface;
public class InterfaceTest implements I1, I2{
    public void m1()
    {
        System.out.println("method m1 from Interface I1");
    }
    public void m2()
    {
        System.out.println("method m2 from Interface I1");
    }
    public void m3()
    {
        System.out.println("method m3 from Interface I2");
    }
    public void m4()
    {
        System.out.println("method m4 from Interface I2");
    }
    public static void main(String[] args)
    {
        InterfaceTest t1 = new InterfaceTest();
        t1.m1();
        t1.m2();
        t1.m3();
        t1.m4();
    }
}
```

Casting:

- Converting one type of information into another type is called casting

  In java casting is classified into 2 types:

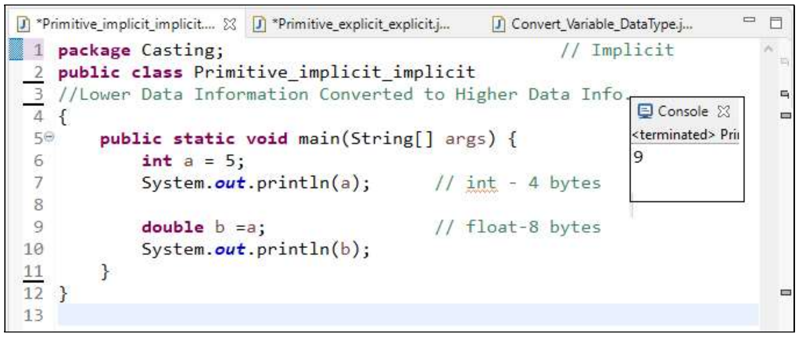
  1. Primitive casting
  2. Non-primitive casting

1. Primitive-casting:

- Converting one data type of information into another data type is called Primitive-casting
- Primitive-casting is classified into 3 types:
  1. implicit casting
  2. explicit casting
  3. Boolean casting

1. Implicit casting:

- Converting lower data type information into higher data type information is called implicit casting.
- Implicit casting is also called widening casting, where memory size goes on increasing.

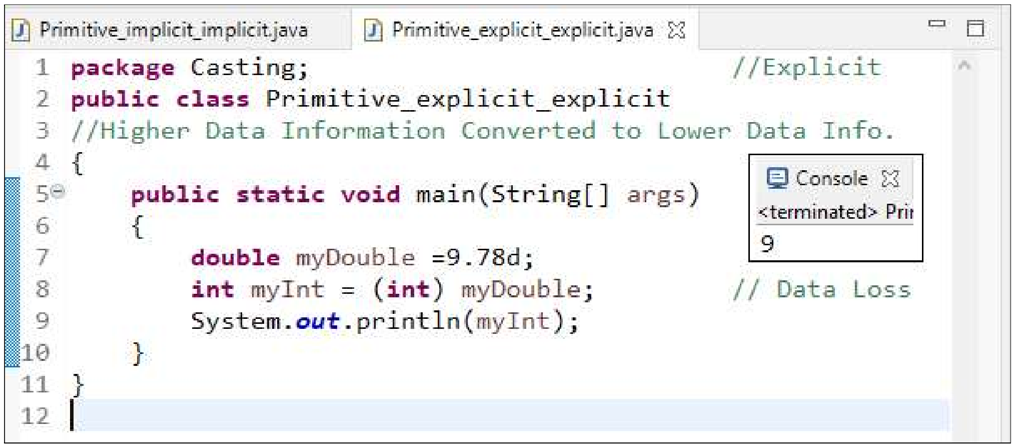


```
package Casting;                              // Implicit
public class Primitive_implicit_implicit
//Lower Data Information Converted to Higher Data Info.
{
	public static void main(String[] args) {
		int a = 5;
		System.out.println(a);        // int - 4 bytes

		double b =a;                  // float-8 bytes
		System.out.println(b);
	}
}
```

```
Console
<terminated> Pri
9
```

2. Explicit casting:

- Converting higher data type information into lower data type information is called explicit casting.
- Explicit casting is also called narrowing casting, where memory size goes on decreasing.
- In explicit casting data may loss takes place



```
package Casting;                              //Explicit
public class Primitive_explicit_explicit
//Higher Data Information Converted to Lower Data Info.
{
	public static void main(String[] args)
	{
		double myDouble =9.78d;
		int myInt = (int) myDouble;           // Data Loss
		System.out.println(myInt);
	}
}
```

```
Console
<terminated> Pri
9
```

3. Boolean casting:

- Boolean casting is considered to be incompatible casting type, because Boolean data type is unique type of data type where information is already predeclared inside it.

```
boolean str = true
```

3. Non-primitive casting
    - converting one type of class into another type of class is called non-primitive casting.

    non-primitive is classified into 2 types:

    1. Up casting (**)
    2. Down casting

1. up casting:

- Assigning subclass property into superclass is called upcasting.
- Before performing upcasting 1st we need to perform inheritance operation.
- After performing inheritance, the property which are present inside superclass comes into subclass
- In the subclass programmer can declare new properties.
- At the time of upcasting operation the properties which are inherited from superclass are only eligible for the upcasting operation.
- The new property which was declared inside subclass are not eligible for upcasting operation.

2. Down casting:

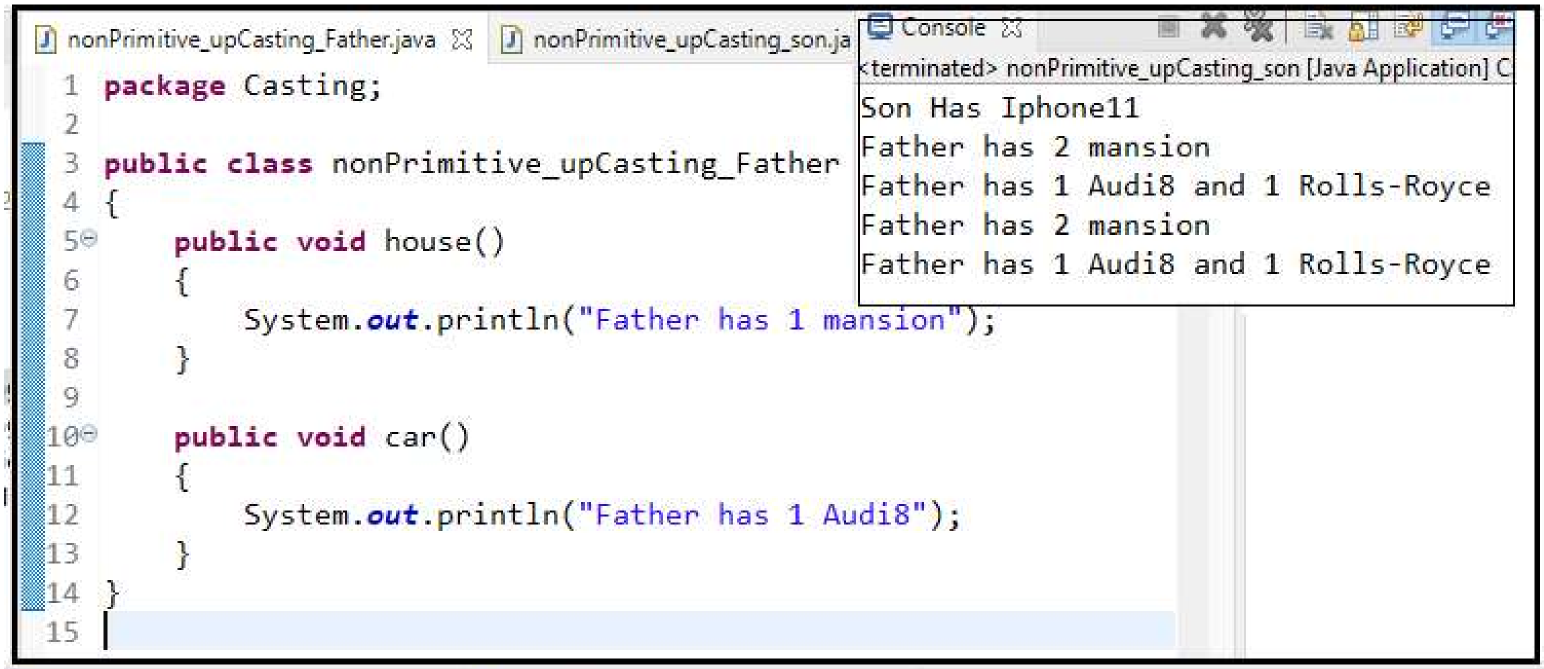
- Assigning superclass property into subclass is called down casting.
- Before performing down casting 1st, we need to perform upcasting.

```
package UpCasting;                              //parent / base / super
public class father
{
        public void money() {

                System.out.println("money: 1L");
        }

        public void car() {
                System.out.println("car:Honda city");
        }

        public void home() {
                System.out.println("home: 2 bhk");
        }

}
```

```
package UpCasting;                              //child / sub class
public class son extends father
{
        public void mobile()
        {
                System.out.println("mobile: samsung");
        }

        public void money()                     //method override
        {
                System.out.println("money: 0.5L");
        }

        public void car()                       //method override
        {
                System.out.println("car:Kia Seltos");
        }
}
```

```
package UpCasting;
public class TestUpCasting{
public static void main(String[] args)
 {
        //create object of sub class provide reference of super class
        father s=new son();   // Superclass ObjectName = new Subclass
        s.money();
        s.car();
        s.home();
}
}
```

Up-Casting

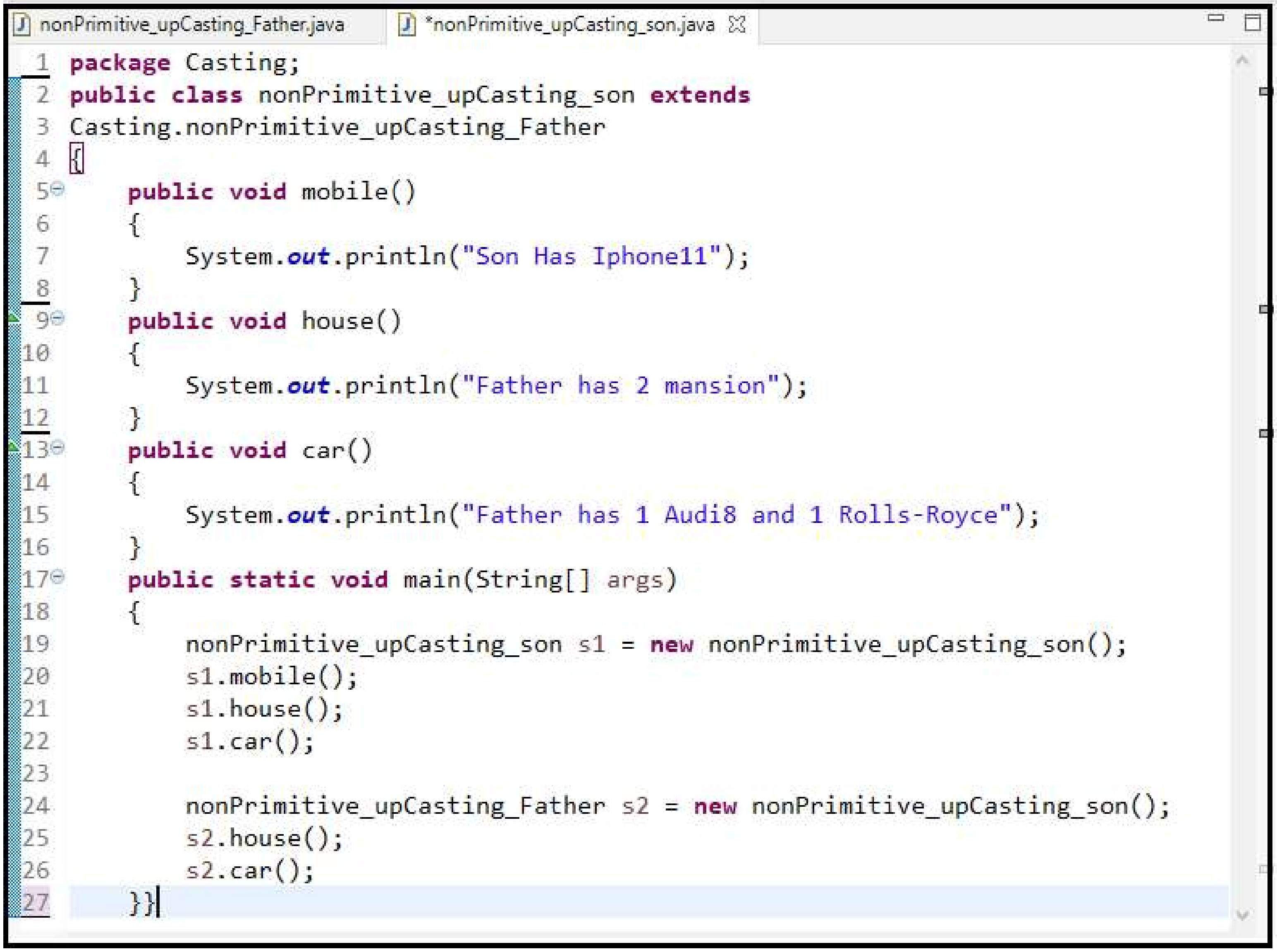


nonPrimitive_upCasting_Father.java   nonPrimitive_upCasting_son.ja

```java
package Casting;

public class nonPrimitive_upCasting_Father
{
	public void house()
	{
		System.out.println("Father has 1 mansion");
	}

	public void car()
	{
		System.out.println("Father has 1 Audi8");
	}
}

```

Console

```
<terminated> nonPrimitive_upCasting_son [Java Application] C
Son Has Iphone11
Father has 2 mansion
Father has 1 Audi8 and 1 Rolls-Royce
Father has 2 mansion
Father has 1 Audi8 and 1 Rolls-Royce
```



nonPrimitive_upCasting_Father.java   *nonPrimitive_upCasting_son.java

```java
package Casting;
public class nonPrimitive_upCasting_son extends
Casting.nonPrimitive_upCasting_Father
{
	public void mobile()
	{
		System.out.println("Son Has Iphone11");
	}
	public void house()
	{
		System.out.println("Father has 2 mansion");
	}
	public void car()
	{
		System.out.println("Father has 1 Audi8 and 1 Rolls-Royce");
	}
	public static void main(String[] args)
	{
		nonPrimitive_upCasting_son s1 = new nonPrimitive_upCasting_son();
		s1.mobile();
		s1.house();
		s1.car();

		nonPrimitive_upCasting_Father s2 = new nonPrimitive_upCasting_son();
		s2.house();
		s2.car();
	}}
```

Abstraction:

- Abstraction is one of the oops principle in java.
- Hiding the implementation code and providing only functionality to the end user is called abstraction.
- The scenario of Abstraction is "if customer is visiting or making use of any application, then he should utilize functionality only & he should not feel any backend code processing"

Generalization:

- Extracting all the important common properties & declaring it in super class (i.e. super interface) & providing implementation/definition according to subclass specification is called Generalization.
- Generalization file can be normal java class or abstract class or Interface, but only Interface is recommended.

```
package Generalization;                    //super interface  --> Generalization file
public interface SimCard
{
        void sms();
        void audioCalling();
        void internet();
}
```

```
package Generalization;
public class Jio  implements SimCard
{
        public void sms()
        {
                System.out.println("sms: 1000");
        }
        public void audioCalling()
        {
                System.out.println("audioCalling: unlimited");
        }
        public void internet()
        {
                System.out.println("internet: 3GB");
        }
        public void newFeatureA()
        {
                System.out.println("newFeature: A");
        }}--------------------------------------
```

```
package Generalization;
public class VI implements SimCard
{
        public void sms()
        {
                System.out.println("sms: 500");
        }
        public void audioCalling()
        {
                System.out.println("audioCalling: 200");
        }
        public void internet()
        {
                System.out.println("internet: 2GB");
        }
        public void newFeatureB() {
                System.out.println("newFeature: B");
        }
}
```

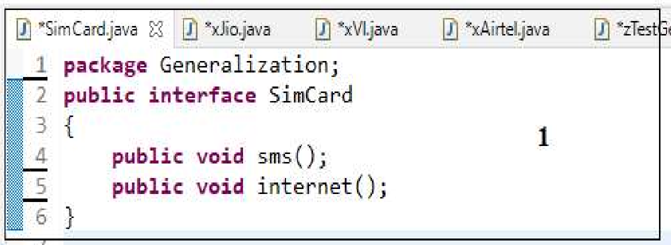
```
package Generalization;
public class Airtel implements SimCard
{
        public void sms()
        {
                System.out.println("sms: 400");
        }
        public void audioCalling()
        {
                System.out.println("audioCalling: 100");
        }
        public void internet()
        {
                System.out.println("internet: 1GB");
        }
        public void newFeatureC()
        {
                System.out.println("newFeature: C");
        }
}
```

Below code is for reference only

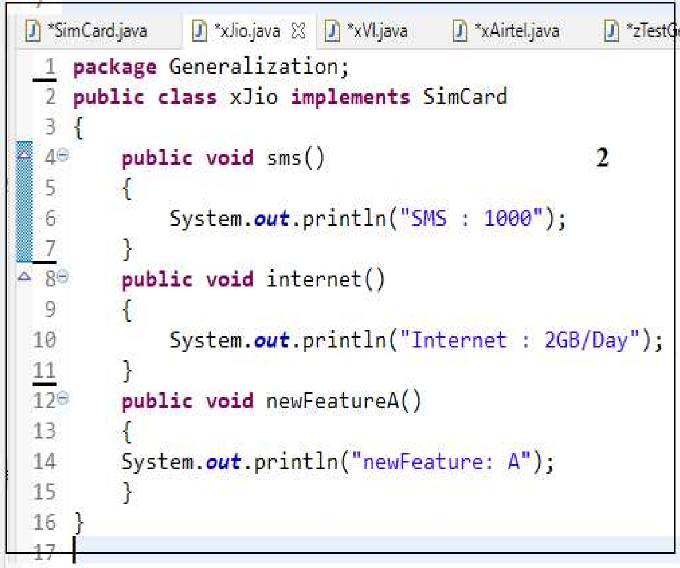
```
package Generalization;
public class zTestGeneralization
{
public static void main(String[] args)
{
System.out.println("---------Properties of Jio------------");
xJio j = new xJio();
j.sms();
j.internet();
j.newFeatureA();

System.out.println("---------Properties of VI------------");
xVI v =new xVI();
v.sms();
v.internet();
v.newFeatureB();
```

```java
System.out.println("---------Properties of Airtel-----------");
xAirtel a = new xAirtel();
a.sms();
a.internet();
a.newFeatureC();
}
}
```

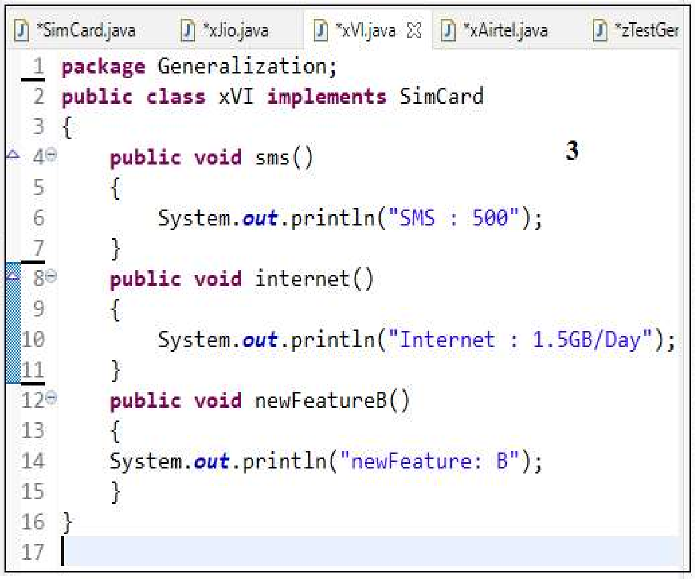
Problem on Generalization



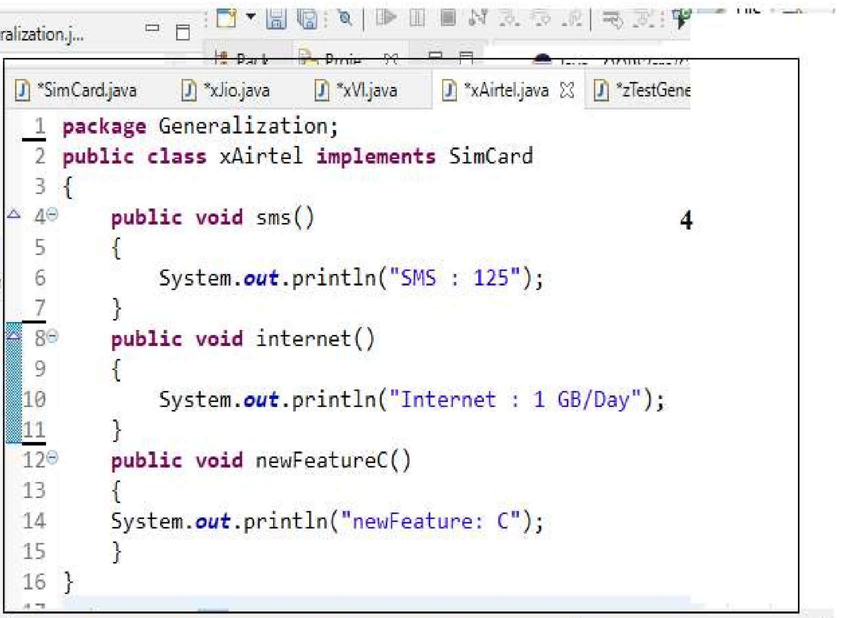
```java
package Generalization;
public interface SimCard
{
    public void sms();
    public void internet();
}
```



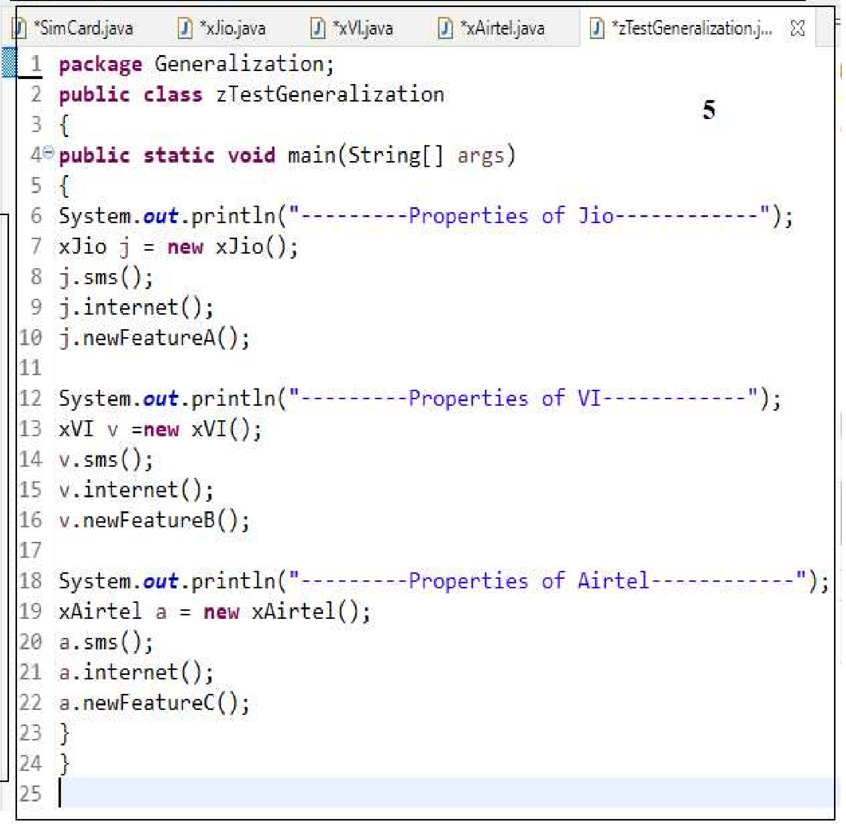
```java
package Generalization;
public class xJio implements SimCard
{
    public void sms()
    {
        System.out.println("SMS : 1000");
    }
    public void internet()
    {
        System.out.println("Internet : 2GB/Day");
    }
    public void newFeatureA()
    {
    System.out.println("newFeature: A");
    }
}
```



```java
package Generalization;
public class xVI implements SimCard
{
    public void sms()
    {
        System.out.println("SMS : 500");
    }
    public void internet()
    {
        System.out.println("Internet : 1.5GB/Day");
    }
    public void newFeatureB()
    {
    System.out.println("newFeature: B");
    }
}
```



```java
package Generalization;
public class xAirtel implements SimCard
{
    public void sms()
    {
        System.out.println("SMS : 125");
    }
    public void internet()
    {
        System.out.println("Internet : 1 GB/Day");
    }
    public void newFeatureC()
    {
    System.out.println("newFeature: C");
    }
}
```



```java
package Generalization;
public class zTestGeneralization
{
public static void main(String[] args)
{
System.out.println("---------Properties of Jio-----------");
xJio j = new xJio();
j.sms();
j.internet();
j.newFeatureA();

System.out.println("---------Properties of VI-----------");
xVI v =new xVI();
v.sms();
v.internet();
v.newFeatureB();

System.out.println("---------Properties of Airtel-----------");
xAirtel a = new xAirtel();
a.sms();
a.internet();
a.newFeatureC();
}
}
```

Use of Arrays

- Array is a data structure used to store the collection of information of same data type
- Arrays are homogenous in nature (i.e., two different data types are not allowed in single defined object)
- Array declaration is need to be done with capacity. (new String [5])
- Arrays are not growable i.e., size is fixed
- Array is an object
- In the array object, indexing start from the Zero (0)

Types of Arrays

1. Single Dimensional Arrays -→ int [ ] ar =new int[5];
2. Multidimensional Arrays → int [ ][ ] ar1 = new int[2][3]; (Rows X Columns)

```
package Array;
public class example1_intArray
{
public static void main(String[] args)
{
        //step1: array declaration
        int []   ar=new int[5];
        //step2: array initialization
        ar[0]=200;
        ar[1]=300;
        ar[2]=400;
        ar[3]=500;
        ar[4]=100;
        //ar[5]=600;

        System.out.println(ar[4]);

        //step3: array usage
        System.out.println(ar[0]);    //200
        System.out.println(ar.length);   //5

        System.out.println("---------print all info from array-----------");

        for(int i=0; i<=4; i++)
        {
                System.out.println(ar[i]);
        }

        for(int i=0; i<=ar.length-1;i++)
        {
                System.out.println(ar[i]);
        }
}
}
```

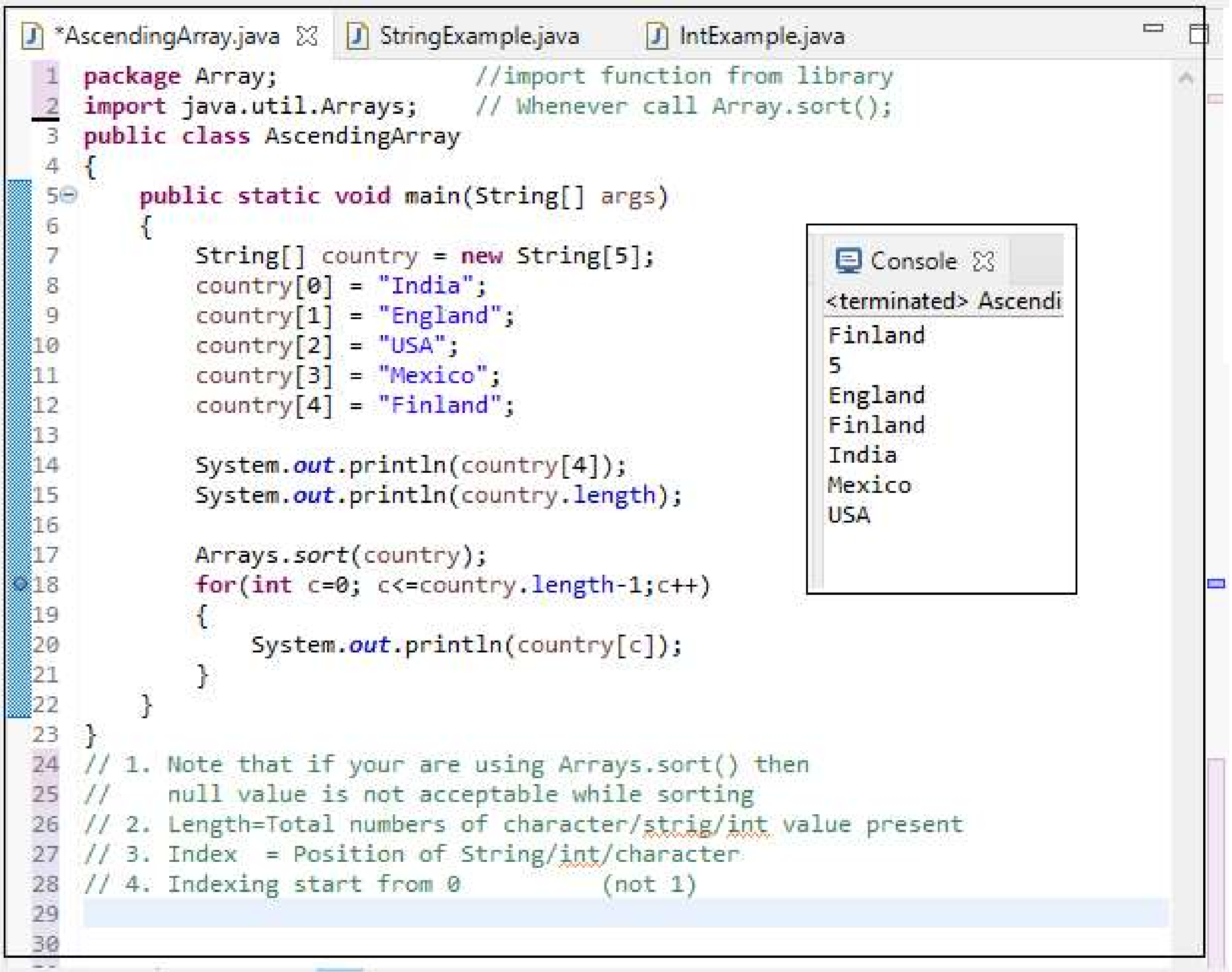
```
package Array;
public class example2_StringArray {
public static void main(String[] args) {

          String [] ar1=new String[4];//create object of string array with size 4
          ar1[0]="mahesh";
          ar1[1]="ramesh";
          ar1[2]="suresh";
          ar1[3]="ganesh";

          System.out.println(ar1[2]);   //suresh
          System.out.println(ar1.length);
          System.out.println("----print all data from array------");

//        for(int i=0; i<=3; i++)
//        {
//                  System.out.println(ar1[i]);
//        }

          for(int i=0; i<=ar1.length-1; i++)
          {
                    System.out.println(ar1[i]);
          }
}
}
```

```
package Array;
public class example3_PrintArrayInReverseOrder
{
public static void main(String[] args)
{
                    int []    ar=new int[5];
                    ar[0]=200;
                    ar[1]=300;
                    ar[2]=400;
                    ar[3]=500;
                    ar[4]=100;

          for(int i=ar.length-1; i>=0; i--)
          {
                    System.out.println(ar[i]);
          }
}
}
```

```
package Array;
import java.util.Arrays;
public class example4_ArraySort
{
public static void main(String[] args) {
                    int []    ar=new int[5];
                    ar[0]=200;  //100
                    ar[1]=300;  //200
                    ar[2]=400;  //300
                    ar[3]=500;  //400
                    ar[4]=100;  //500

System.out.println("-----print original info----");
for(int i=0; i<=ar.length-1; i++)
{
          System.out.println(ar[i]);
}
System.out.println("-----print info in ascending order----");
Arrays.sort(ar);

for(int i=0; i<=ar.length-1; i++)
{
          System.out.println(ar[i]);
}
System.out.println("-----print info in descending order----");
for(int i=ar.length-1; i>=0; i--) {
          System.out.println(ar[i]);
}
}
}
```

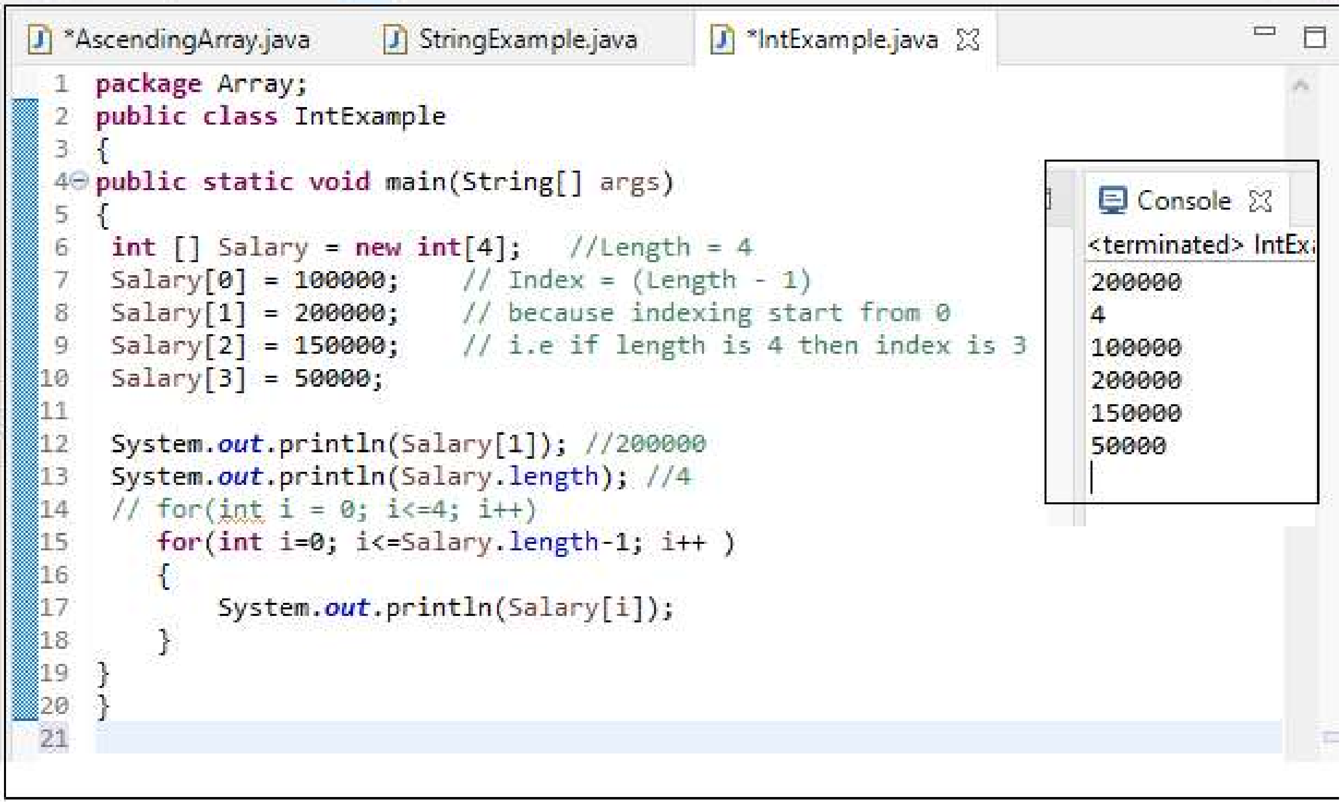
```
package Array;
import java.util.Arrays;
public class example5_StringArray_Sorting
{
public static void main(String[] args)
 {
          String [] ar1=new String[4];
          ar1[0]="mahesh";
          ar1[1]="ramesh";
          ar1[2]="suresh";
          ar1[3]="ganesh";
          System.out.println("----print original data------");
          for(int i=0; i<=ar1.length-1; i++)
          {
                    System.out.println(ar1[i]);
          }
          System.out.println("----print string info in alphabetical order------");
          Arrays.sort(ar1);
          for(int i=0; i<=ar1.length-1; i++)
          {
                    System.out.println(ar1[i]);
          }
}
}
```

package Array;



```java
package Array;            //import function from library
import java.util.Arrays;    // Whenever call Array.sort();
public class AscendingArray
{
    public static void main(String[] args)
    {
        String[] country = new String[5];
        country[0] = "India";
        country[1] = "England";
        country[2] = "USA";
        country[3] = "Mexico";
        country[4] = "Finland";

        System.out.println(country[4]);
        System.out.println(country.length);

        Arrays.sort(country);
        for(int c=0; c<=country.length-1;c++)
        {
            System.out.println(country[c]);
        }
    }
}
// 1. Note that if your are using Arrays.sort() then
//    null value is not acceptable while sorting
// 2. Length=Total numbers of character/strig/int value present
// 3. Index  = Position of String/int/character
// 4. Indexing start from 0          (not 1)
```

```
<terminated> Ascendi
Finland
5
England
Finland
India
Mexico
USA
```



```java
package Array;
public class IntExample
{
public static void main(String[] args)
{
 int [] Salary = new int[4];    //Length = 4
 Salary[0] = 100000;     // Index = (Length - 1)
 Salary[1] = 200000;     // because indexing start from 0
 Salary[2] = 150000;     // i.e if length is 4 then index is 3
 Salary[3] = 50000;

 System.out.println(Salary[1]); //200000
 System.out.println(Salary.length); //4
 // for(int i = 0; i<=4; i++)
    for(int i=0; i<=Salary.length-1; i++ )
    {
        System.out.println(Salary[i]);
    }
}
}
```

```
<terminated> IntEx
200000
4
100000
200000
150000
50000
```

```
public class example6_intArray_declarationInitialization_in_singleStep {
public static void main(String[] args) {

        int [] ar1= {10,50,40,30,20};

        System.out.println(ar1.length);

        System.out.println("-----print all info from int array-------");
        for(int i=0; i<=ar1.length-1; i++)
        {
                System.out.println(ar1[i]);
        }
}
}
```

```
package Array;
public class example7_StringArray_declarationInitialization_in_singleStep {
public static void main(String[] args) {

        String [] ar= {"mahesh","ramesh", "suresh","ganesh"};

        System.out.println(ar.length);  //4
        System.out.println("---print all info from string array----");
        for(int i=0; i<=ar.length-1; i++) {
                System.out.println(ar[i]);
        }
}
}
```

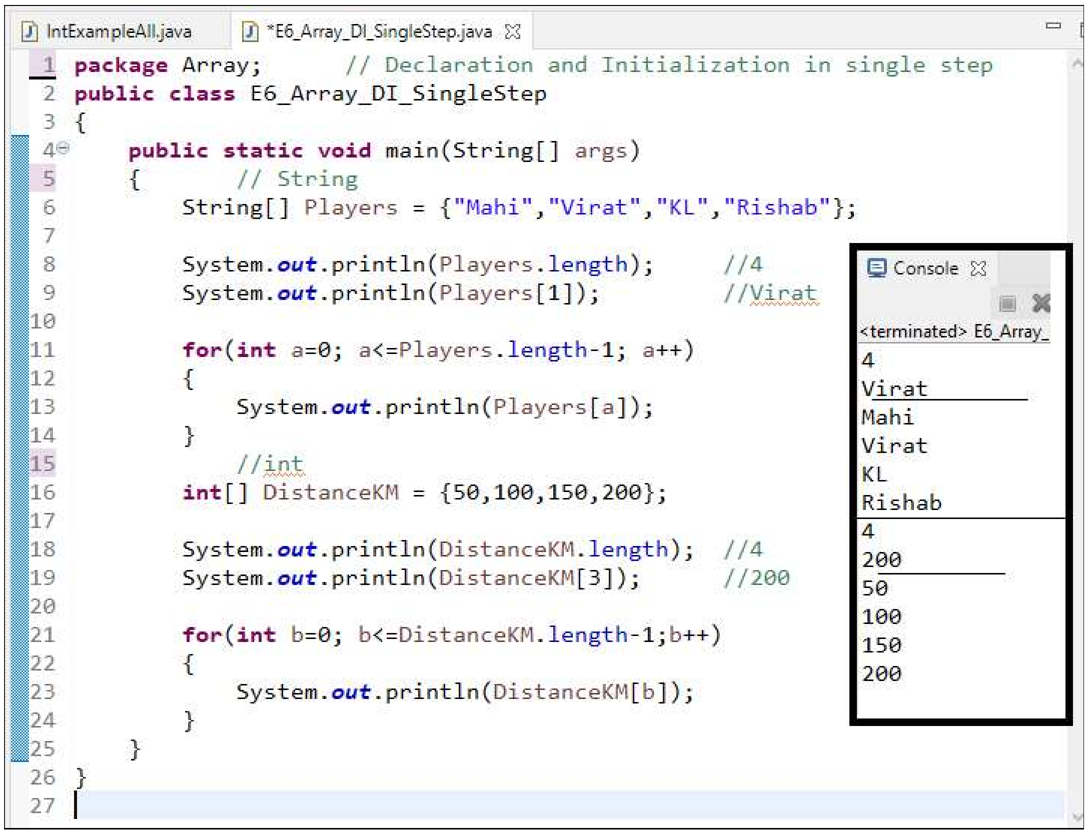
(Multidimensional Arrays)

```
package Array;
public class example8 {
public static void main(String[] args) {
        //    0  1  2
        //0 10 20 30
        //1 40 50 60

        int [][] ar=new int[2][3];
        ar[0][0]=10;
        ar[0][1]=20;
        ar[0][2]=30;
        ar[1][0]=40;
        ar[1][1]=50;
        ar[1][2]=60;

        System.out.println(ar.length);   //2
        System.out.println(ar[0][2]);   //60

        System.out.println("-----print array--------");

        //outer for loop for rows
        //        //2<=1  2
```

```
          for(int i=0; i<=1; i++)
          {
                    //inner for loop for cols
                                                  //3<=2 3
                    for(int j=0; j<=2; j++)
                    {                                       // 1   2
                              System.out.print(ar[i][j]+" ");
                    }
                    System.out.println();
          }
          //0 10 20 30
          //1 40 50 60
}
}
```

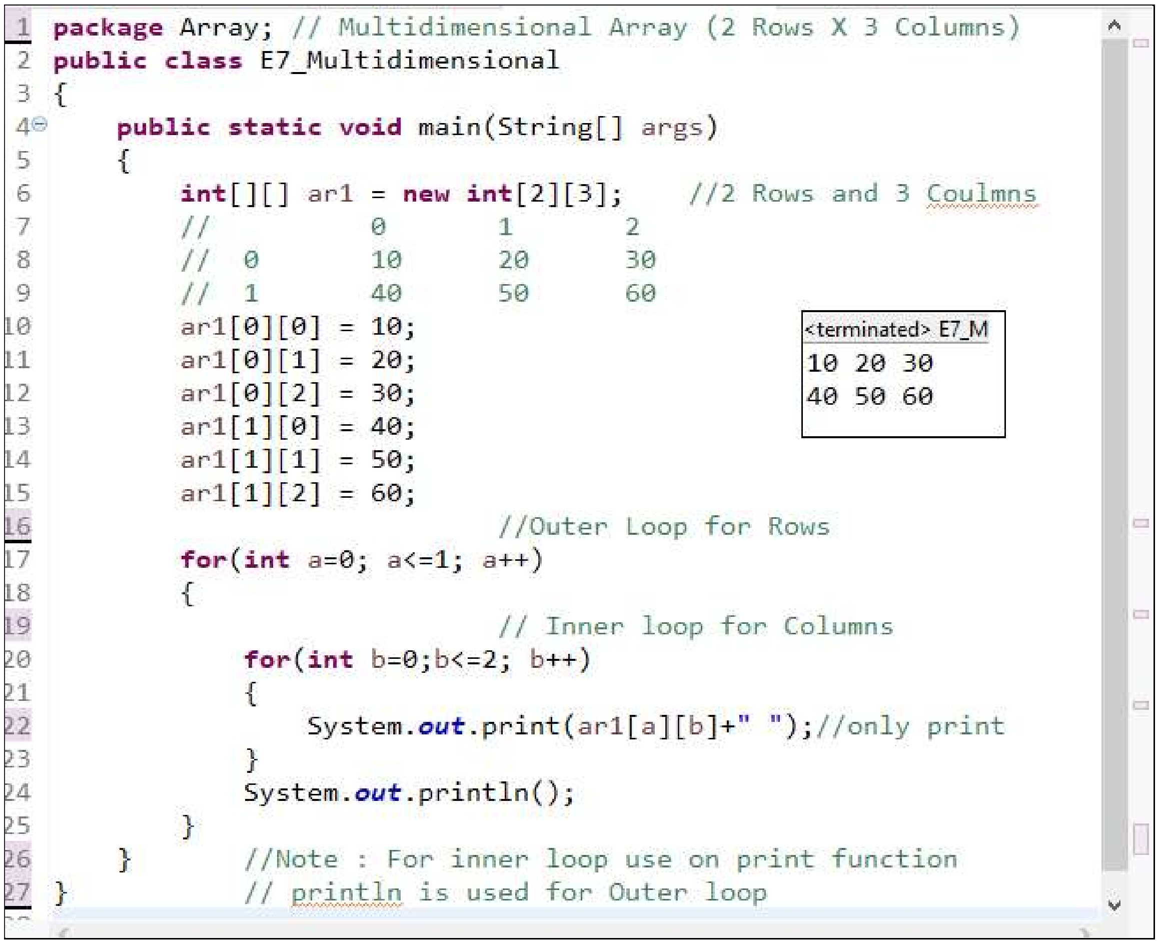
```
package Array;
public class example9 {
public static void main(String[] args) {
          //  0  1  2
          //0 10 20 30
          //1 40 50 60

          int [][] ar= {{10,20,30},{40,50,60}};

          for (int i = 0; i <=1; i++) {
                    for (int j = 0; j <=2; j++)
                    {
                              System.out.print(ar[i][j]+" ");
                    }
                    System.out.println();
          }
}
}
```

(Array with Declaration and Initialization)



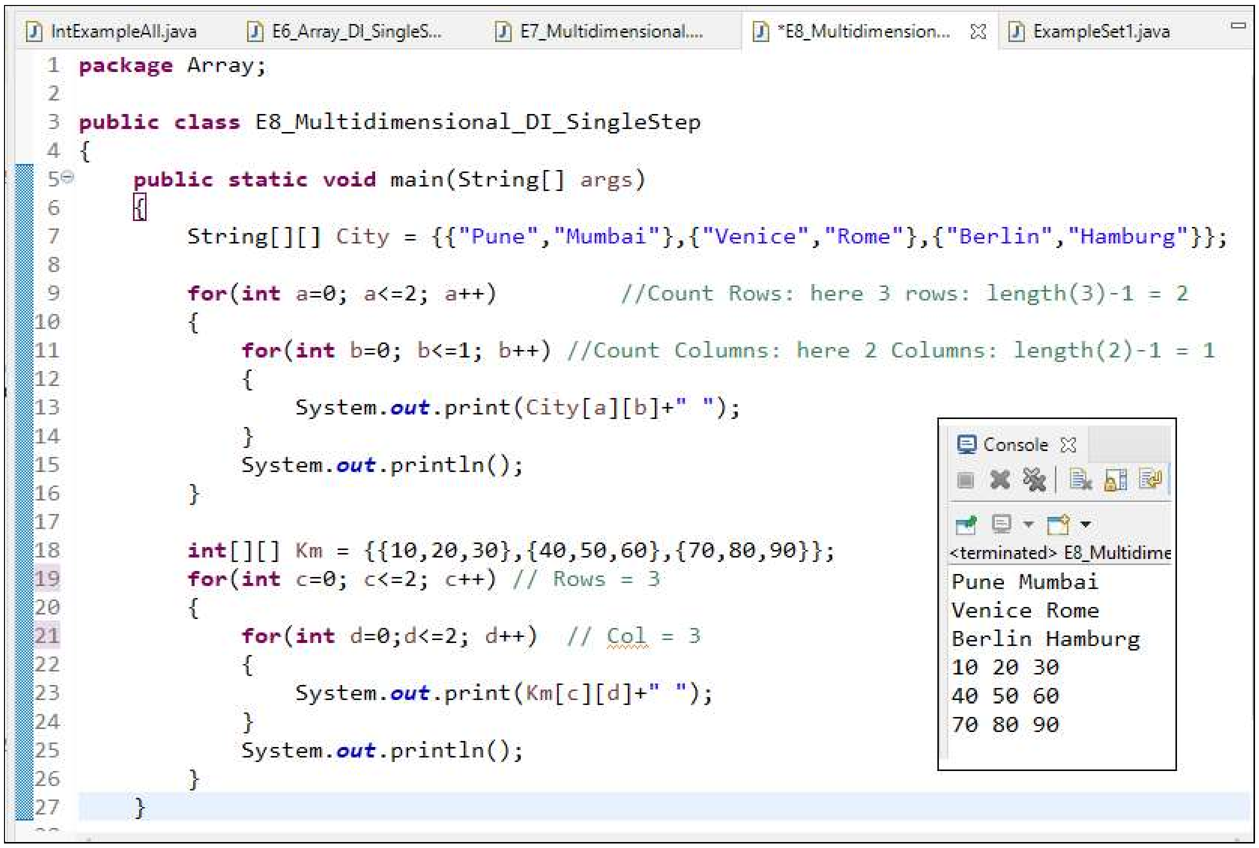
```java
package Array;      // Declaration and Initialization in single step
public class E6_Array_DI_SingleStep
{
    public static void main(String[] args)
    {       // String
        String[] Players = {"Mahi","Virat","KL","Rishab"};

        System.out.println(Players.length);     //4
        System.out.println(Players[1]);         //Virat

        for(int a=0; a<=Players.length-1; a++)
        {
            System.out.println(Players[a]);
        }
            //int
        int[] DistanceKM = {50,100,150,200};

        System.out.println(DistanceKM.length);  //4
        System.out.println(DistanceKM[3]);      //200

        for(int b=0; b<=DistanceKM.length-1;b++)
        {
            System.out.println(DistanceKM[b]);
        }
    }
}
```

```
<terminated> E6_Array_
4
Virat
Mahi
Virat
KL
Rishab
4
200
50
100
150
200
```



```java
package Array; // Multidimensional Array (2 Rows X 3 Columns)
public class E7_Multidimensional
{
    public static void main(String[] args)
    {
        int[][] ar1 = new int[2][3];    //2 Rows and 3 Coulmns
        //          0       1       2
        //  0       10      20      30
        //  1       40      50      60
        ar1[0][0] = 10;
        ar1[0][1] = 20;
        ar1[0][2] = 30;
        ar1[1][0] = 40;
        ar1[1][1] = 50;
        ar1[1][2] = 60;
                        //Outer Loop for Rows
        for(int a=0; a<=1; a++)
        {
                        // Inner loop for Columns
            for(int b=0;b<=2; b++)
            {
                System.out.print(ar1[a][b]+" ");//only print
            }
            System.out.println();
        }
    }       //Note : For inner loop use on print function
}           // println is used for Outer loop
```

```
<terminated> E7_M
10 20 30
40 50 60
```



```java
package Array;

public class E8_Multidimensional_DI_SingleStep
{
    public static void main(String[] args)
    {
        String[][] City = {{"Pune","Mumbai"},{"Venice","Rome"},{"Berlin","Hamburg"}};

        for(int a=0; a<=2; a++)         //Count Rows: here 3 rows: length(3)-1 = 2
        {
            for(int b=0; b<=1; b++) //Count Columns: here 2 Columns: length(2)-1 = 1
            {
                System.out.print(City[a][b]+" ");
            }
            System.out.println();
        }

        int[][] Km = {{10,20,30},{40,50,60},{70,80,90}};
        for(int c=0; c<=2; c++) // Rows = 3
        {
            for(int d=0;d<=2; d++)  // Col = 3
            {
                System.out.print(Km[c][d]+" ");
            }
            System.out.println();
        }
    }
```

```
Pune Mumbai
Venice Rome
Berlin Hamburg
10 20 30
40 50 60
70 80 90
```

String class:

1. String is non-primitive data type; memory size is not fixed.
2. String is used to store collection of characters.
3. String is an inbuilt class present inside "java.lang" package.
4. String class is final class can't be inherited to other classes.
5. At the time of String declaration, initialization, object creation takes place.
6. String objects are immutable in nature/can't be change.
7. Object creation of String can be done in 2 ways:
    1. without using new keyword
    2. Using new keyword
8. String objects are going to get stored inside String pool area which is present inside heap area.

String pool area:

- It is used to store String objects.
- It is classified into 2 areas:
    1. Constant pool area
    2. Non-constant pool area.

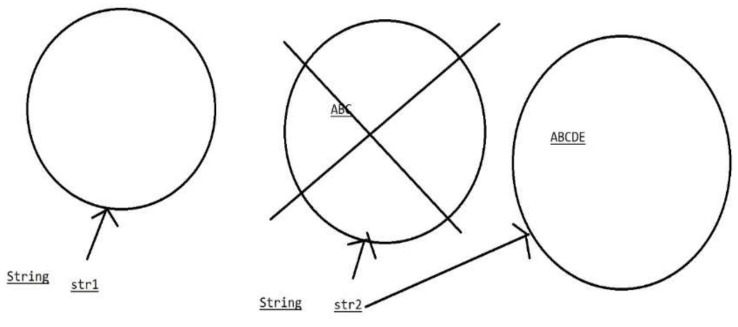
1. Constant pool area:

1. During object creation time if you don't make use of new keyword then object creation takes place inside constant pool area.
2. Duplicate objects are not allowed inside constant pool area.

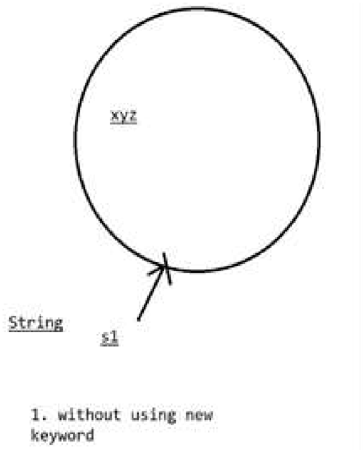
2. 0Non-constant pool area:

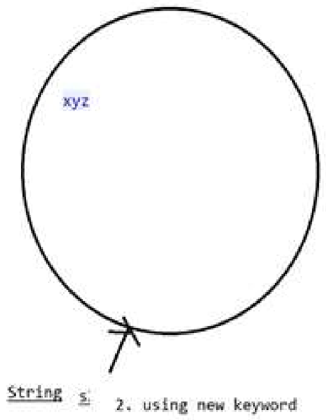
1. During object creation time if you make use of new keyword then object creation takes place inside non-constant pool area.
2. Duplicate objects are allowed inside non-constant pool area.

```
package StringClass;
final public class sample1
{
public static void main(String[] args)
 {
        //System.out.println();
        String str1;
        String str2="ABC";
        System.out.println(str2);

        str2=str2+"DE";      //ABCDE
        System.out.println(str2);
}
}
```

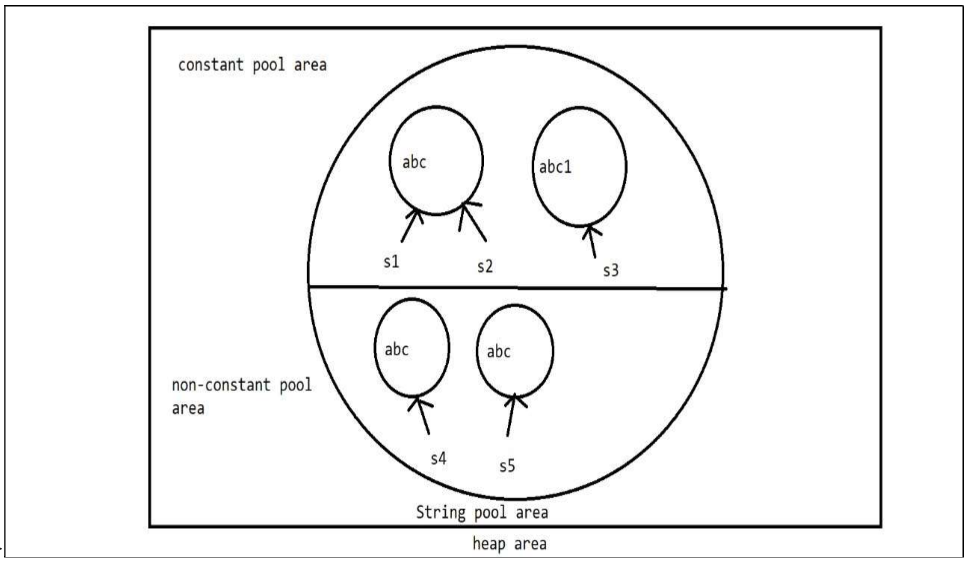


```
package StringClass;
public class sample2
{
        public static void main(String[] args) {

                //object creation of string
                //1. without using new keyword
                String s1="xyz";

                //2. using new keyword
                String s2=new String("xyz");
        }
}
```





```
package StringClass;
public class sample3 {
public static void main(String[] args) {

        //without using new keyword ---> constant pool area
        String  s1="abc";
        String s2="abc";
        String s3="abc1";

        // using new keyword---> non-constant pool area
        String s4=new String("abc");
        String s5=new String("abc");

        System.out.println(s1==s2);          //true
        System.out.println(s1==s3);          //false
        System.out.println(s1==s4);          // false
        System.out.println(s4==s5);          //false
}
```



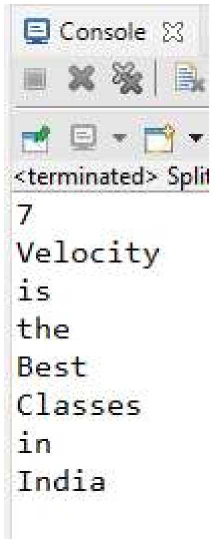
}

```java
package String_Class;
public class String_Class_Methods
{
public static void main(String[] args)
{
        //    1. length()
        //    Will Print length of the string
        String Name ="Mr.Vaibhav Yendole";           // Length  = 18
        System.out.println(Name.length());
        // Output : 18 (Space also considered while counting)


        // 2. toUpperCase()
        // 3. toLowerCase()
        // Print the String in Uppercase or Lower Case
        String City = "Berlin";            // Valid up to execution only
        System.out.println(City.toUpperCase()); // Output = BERLIN

        City = City.toUpperCase();  // Assigned value of UpperCase
        System.out.println(City);                    // Output = BERLIN

        System.out.println(City.toLowerCase());      // Output = berlin

        City = City.toLowerCase();  // Assigned value of LowerCase
        System.out.println(City);                    // Output = berlin


        // 4. equals()
        // If two string values are exactly equal then print true
        // otherwise false. ( characters are case sensitive)
        String E1 = "Velocity";
        String E2 = "VELOCITY";
        String E3 = "VELOCITY";
        System.out.println(E1.equals(E2));           // Output = false
        System.out.println(E1.equals(E3));           // Output = true


        // 5. equalsIgnoreCase();
        // It will ignore case sensitive property only for equalsIgnoreCase();
        String E4 = "Velocity";
        String E5 = "VELOCITY";
        System.out.println(E4.equalsIgnoreCase(E5));      // Output= true


        // 6. contains()
        // Some sequential character also consist in other string
        String c1 = "ManchesterUnited";
        String c2 = "United";
        String c3 = "Unitedly";
        System.out.println(c1.contains(c2));   // Output = true
        System.out.println(c2.contains(c1));   // Output = false
        System.out.println(c1.contains(c3));   // Output = false
```

```java
// 7. isEmpty();
// If the string has empty (not even space) then print true
String E7 ="";
String E8 = "Denver";
System.out.println(E7.isEmpty());           // Output = true
System.out.println(E8.isEmpty());           // Output = false


// 8. charAt(int index);                     //charAt(3)
// Will Print the single character of mentioned index value
String C4 = "Christiano";
System.out.println(C4.charAt(0));           // Output = C
System.out.println(C4.charAt(1));           // Output = h
System.out.println(C4.charAt(2));           // Output = r


// 9. startWith();
// 10. endWith();
// Will print true if start value or end value matched with string value
String s1 = "Apache220";
System.out.println(s1.startsWith("Apa"));   // Output = true
System.out.println(s1.startsWith("bpa"));   // Output = false
System.out.println(s1.endsWith("220"));     // Output = true
System.out.println(s1.endsWith("120"));     // Output = false


// 11. substring() or substring(start int ,end int)
// Will Print according to position of index mentioned
String S2 = "CSKvsMI";
System.out.println(S2.substring(5));        // Output = MI
// Note : Here 5 is the index and also start point i.e
// 5th position of index is considered in output
System.out.println(S2.substring(0, 3));          // Output = CSK
System.out.println(S2.substring(3, 5));          // Output = vs
// Note that end point = (n+1)


// 12. concat();
// Combining two or more than two strings
String C1 = "Ganpati";
String C2 = "Bappa";
String C3 = "Maorya";
System.out.println(C1.concat(C3.concat(C3)));
//Output = GanpatiMaoryaMaorya

System.out.println(C1+" "+C2+" "+C3);
//Output = Ganpati Bappa Maorya
```

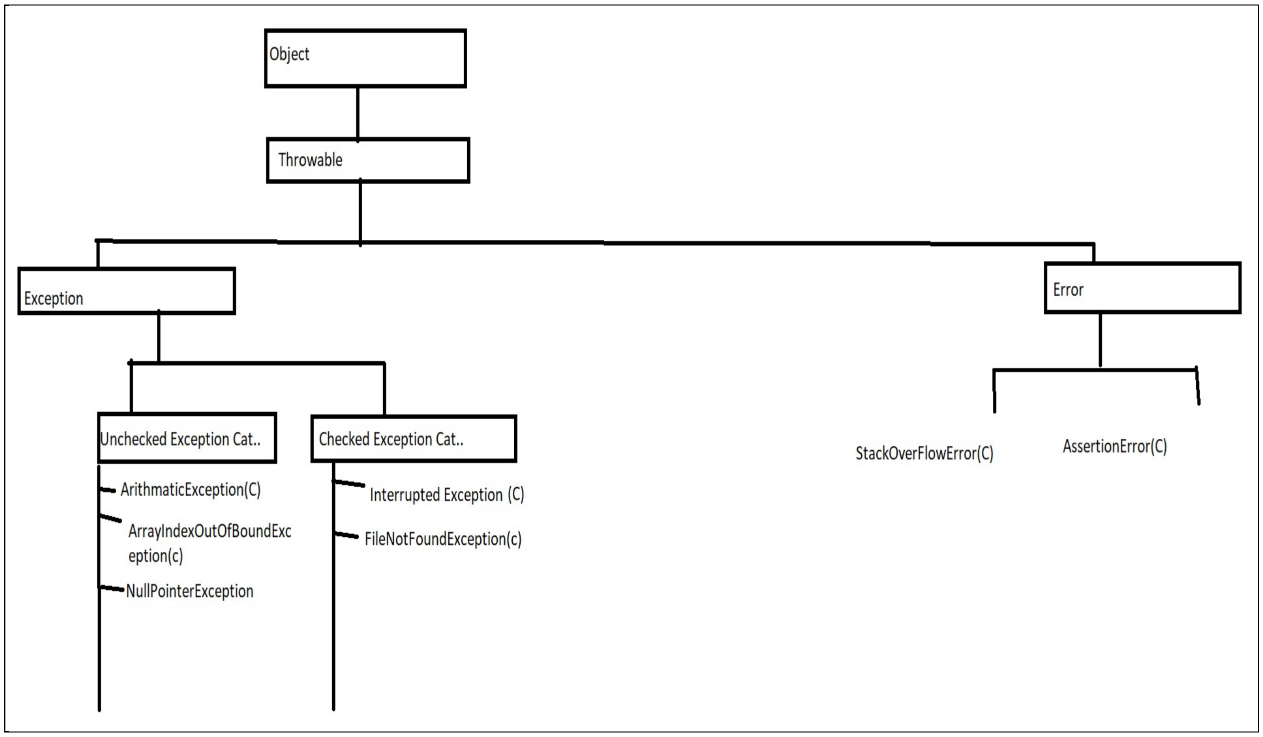
```
        // 13. indexOf()
        // 14. lastIndexOf()
        // indexOf(" ") --> print index of character Count from start
        // lastIndexOf(" ")--> print lastIndexOf Count from End Side
        String inf = "infinity";
        System.out.println(inf.indexOf("i"));           // output = 0
        System.out.println(inf.lastIndexOf("i"));       // output = 5


        // 15. replace()
        // replace()--> replace the String/word exist in current string
        String R1 = “Learn Java”;
        System.out.println(R1.replace(“Java”, “Selenium”));
        // Output = Learn Selenium

        System.out.println(R1.replace(“Learn Java”,”Learn Java by Sanjay
        Sir”));
        // Output = Learn Java by Sanjay Sir
    }
}
```

```
        // 15. split()

        // Use for loop and array for executing.

        // Use to split the Complete String by specific Word

package String_Class;

public class Split_Function
{
        public static void main(String[] args)
        {
                String Study = "Velocity is the Best Classes in India";
                String[] Str = Study.split(" ");        //split by space
                System.out.println(Str.length);

                for(int i=0; i<=Str.length-1; i++)
                {
                        System.out.println(Str[i]);
                }
        }
}
```



```
7
Velocity
is
the
Best
Classes
in
India
```

• Exception Handling



- Unexpected event - Raining on the road, No internet connectivity
- Exception – problem
- Error – is lack of resources
- Exception – Unexpected event

An Exception is an unexpected event, which occurs during execution of a JAVA program, that disrupt normal flow of the program

----

Example of exception

Example 1 - ArithmeticException

```
package ExceptionHandleing;

public class test1 {
	public static void main(String[] args) {

		System.out.println("1");
		System.out.println("2");
		System.out.println("3");
		System.out.println("4");
		System.out.println(100/0); // here is the issue
		System.out.println("6");
		System.out.println("7");
		System.out.println("8");
	}
}
```

Output :

```
1
2
3
4
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at ExceptionHandleing.test1.main(test1.java:12)
```

Example 2 - ArrayIndexOutOfBoundsException

```
package ExceptionHandleing;
public class test2 {

	public static void main(String[] args) {
		int[] a = {11,12,13,14};

		System.out.println(a[4]);
	}
}
```

Output

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 4
	at ExceptionHandleing.test2.main(test2.java:11)
```

Exception Handling -

Exception handling is process of handling an unexpected event causing an abnormal termination of JAVA program, in such way that program will execute normally

Two ways of Handling an Exception

- Try-catch- finally (unchecked exception)
- Throws keyword (checked exception)

Try-catch- finally

```
-	package ExceptionHandleing;
-
-	public class test1 {
-		public static void main(String[] args) {
-
-			System.out.println("1");
-			System.out.println("2");
-			System.out.println("3");
-			System.out.println("4");
-
-			try {
-			System.out.println(100/0); // risky code line
-			}
-			catch (ArithmeticException messge) {
-				System.out.println("here is the risk and exception is coming");
-                       }
-                       System.out.println("6");
-                       System.out.println("7");
-                       System.out.println("8");
-               }
-   }
```

Output

```
1
2
3
4
here is the risk and exception is coming
6
7
8
```

Try-catch- finally

- Inside try block will write risky code which may cause an exception
- In the catch block we will write the code which can tell us to bypass the situation on which we got an exception
- Only that particular catch will get execute which consist of particular exception
- No matter what finally will always execute
- Purpose of having finally is to closed all the secured access which are given to the script at the start
- It is not mandatory to use finally always

Throws keyword

- Only use to handle checked category exception
- We will write throws keyword beside main method with their exception name

Use of Throws with thread.sleep example

```
package ExceptionHandleing;
public class test1 {
        public static void main(String[] args) throws InterruptedException  {
                System.out.println("1");
                System.out.println("2");
                System.out.println("3");
                System.out.println("4");
                Thread.sleep(5000);  /// InterruptedException
                System.out.println("6");
                System.out.println("7");
                System.out.println("8");

        }}
```

Output :

```
1
2
3
4
6
7
8
```

RefVariable.getMessage() is used to define reason for an exception

Example

```
package ExceptionHandleing;

public class test1 {


          public static void main(String[] args) throws InterruptedException  {
                    System.out.println("1");
                    System.out.println("2");
                    System.out.println("3");
                    System.out.println("4");
                    Thread.sleep(5000);  /// InterruptedException
                    try {
                    System.out.println(100/0); // risky code line
                    }
                    catch (Exception messge) {
                              System.out.println("here is the risk and exception is coming "  +
messge.getMessage()  );

                    }

                    ////////drive logout nahi hua hey
                    finally {
                              /// mera google drive logout kar do
                              System.out.println("finally will alys run no matter what ");
                    }

                    System.out.println("6");
                    System.out.println("7");
                    System.out.println("8");

          }
}
```