In this chapter, we will be learning about of inheritance in Java and the types of inheritance and with help of example.

Inheritance is one of the key features of OOP that allows us to create a new class from an existing class. In Java, it is possible to inherit attributes and methods from one class to another.

The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class/Base Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Why and when Importance of Java inheritance

Implementation of inheritance in Java provides the following benefits:

1) Inheritance minimizes the complexity of a code by minimizing duplicate code. If the same code has to be used by another class, it can simply be inherited from that class to its sub-class. Hence, the code is better organized.
2) It is useful for code reusability, reuse attributes and methods of an existing class when you create a new class.
3) The efficiency of execution of a code increases as the code is organized in a simpler form.
4) The concept of polymorphism can be used along with inheritance.
   Note: the ***extends*** keyword is used to perform inheritance in Java. If you don't want other classes to inherit from a class, use the ***final*** keyword:
   The basic syntax is

```
class superclass {

…….

}

class subclass extends superclass
{

……
……

}
```

```
final class superclass {

        ……

}
```

Types of Java Inheritance

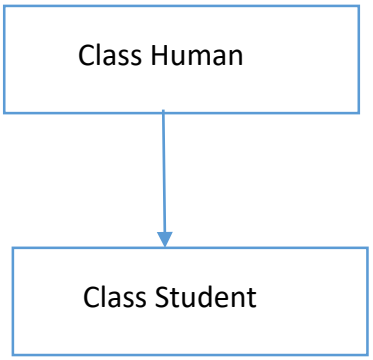
The different types of inheritance are observed in Java:

1. Single level inheritance
2. Multi-level Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

1) Single level inheritance

As the name suggests, this type of inheritance occurs for only a single class. Only one class is derived from the parent class.

The flow diagram of a single inheritance is shown below:



Constructors and Inheritance

constructor of sub class is invoked when we create the object of subclass, it by default invokes the default constructor of super class. Hence, in inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the super keyword, but it should be first statement in a constructor. The super keyword refers to the superclass, immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent is not permitted.

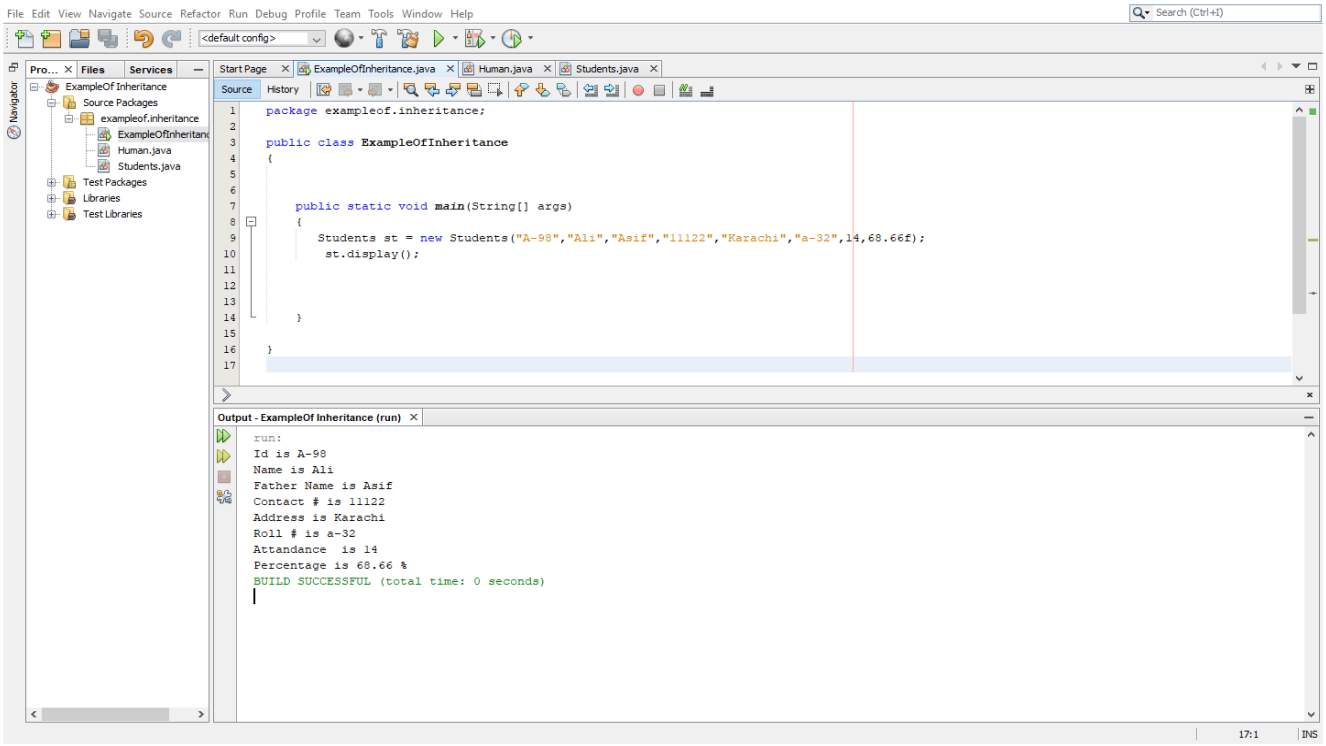
Example of Single Inheritance in java

```
package exampleof.inheritance;

public class Human {
  private String Id;
  private String Name;
  private String FatherName;
  private String Contact;
  private String Address;

  public Human(String id,String name, String fname,String contact,String address)
  {
    Id = id;
    Name = name;
    FatherName = fname;
    Contact = contact;
    Address = address;
  }

  public void display()
  {
    System.out.println("Id is "+Id);
    System.out.println("Name is "+Name);
    System.out.println("Father Name is "+FatherName);
    System.out.println("Contact # is "+Contact);
    System.out.println("Address is "+Address);

  }

}
```

```
package exampleof.inheritance;

public class Students extends Human {
  private String Roll;
  private int Attandance;
  private float Percentage;


  public Students(String i,String n,String f,String c,String a,String roll,int attandance,float per)
  {
    super(i,n,f,c,a);
    Roll =roll;
    Attandance = attandance;
    Percentage = per;
  }

  public void display()
  {
    super.display();
    System.out.println("Roll # is "+Roll);
    System.out.println("Attandance  is "+Attandance);
    System.out.println("Percentage is "+Percentage +" %");

  }
}
```

```
package exampleof.inheritance;
public class ExampleOfInheritance
{
  public static void main(String[] args)
  {
    Students st = new Students("A-98","Ali","Asif","11122","Karachi","a-32",14,68.66f);
     st.display();
```

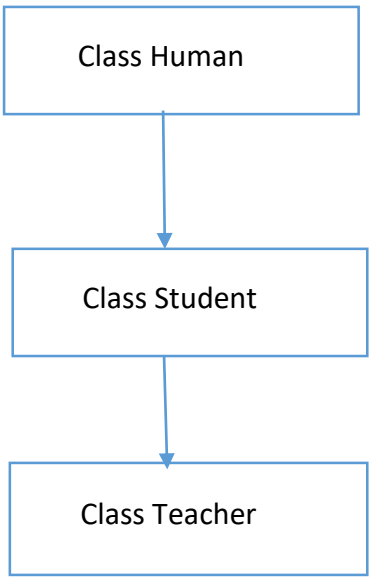
```
    }
}
```



2) Multi-level Inheritance

The multi-level inheritance includes the involvement of at least two or more than two classes. One class inherits the features from a parent class and the newly created sub-class becomes the base class for another new class.

A flow diagram of multi-level inheritance



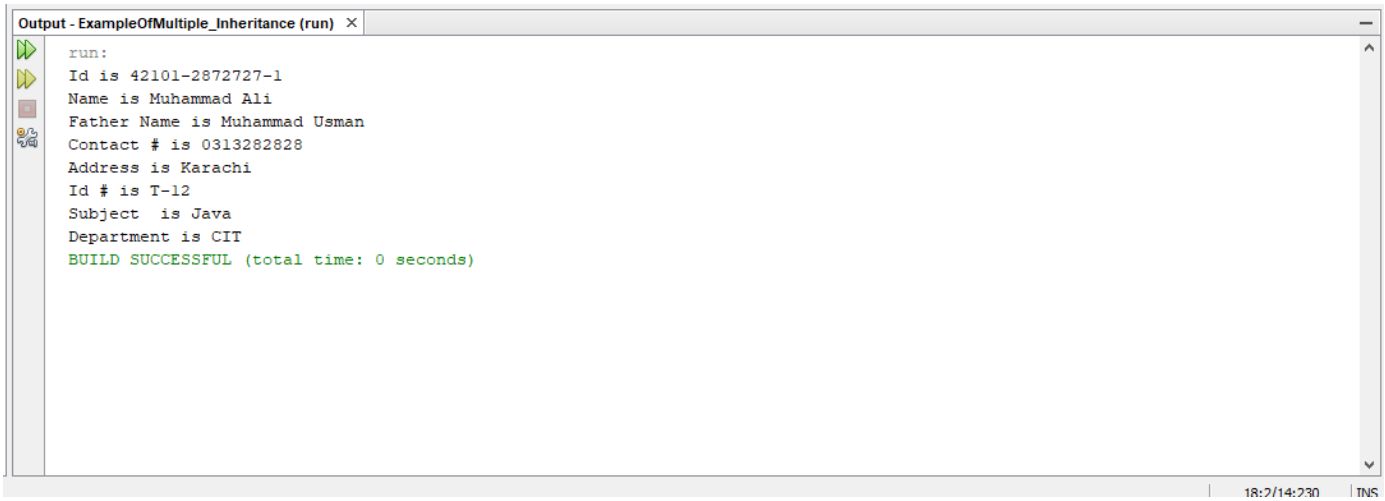
```
package exampleofmultilevel_inheritance;

public class Human {

   private String Id;
   private String Name;
   private String FatherName;
   private String Contact;
   private String Address;

  public Human()
  {
     Id = "42101-2872727-1";
     Name = "Muhammad Ali";
     FatherName = "Muhammad Usman";
     Contact = "0313282828";
     Address = "Karachi";
  }

  public void HumanDisplay()
  {
     System.out.println("Id is "+Id);
     System.out.println("Name is "+Name);
     System.out.println("Father Name is "+FatherName);
     System.out.println("Contact # is "+Contact);
     System.out.println("Address is "+Address);

  }

}
```

```
package exampleofmultilevel_inheritance;

public class Students extends Human{
   private String Roll;
   private int Attandance;
   private float Percentage;

     public Students()
   {

       Roll ="M-172";
       Attandance = 23;
       Percentage = 67;
```

```
    }

    public void StudentDisplay()
    {

      System.out.println("Roll # is "+Roll);
      System.out.println("Attandance  is "+Attandance);
      System.out.println("Percentage is "+Percentage +" %");

    }
}
```

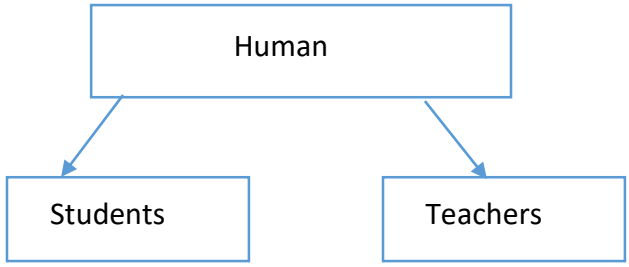
```
package exampleofmultilevel_inheritance;
public class Teacher extends Students
{
    private String Id;
    private String Subject;
    private String Department;

    public Teacher()
    {

        Id ="T-12";
        Subject = "Java";
        Department = "CIT";
    }

    public void TeacherDisplay()
    {

      System.out.println("Id # is "+Id);
      System.out.println("Subject  is "+Subject);
      System.out.println("Department is "+Department);
    }
}
```

```
public class ExampleOfMultiple_Inheritance
{

    public static void main(String[] args)
    {

       Teacher   objTea = new Teacher();
       objTea.HumanDisplay();
       objTea.TeacherDisplay();

    }
}
```



3) Hierarchical Inheritance

The type of inheritance where many subclasses inherit from one single class is known as Hierarchical Inheritance.




```
package exampleof. Hierarchical _inheritance;
public class Human
{
    private String Id;
    private String Name;
    private String FatherName;
    private String Contact;
    private String Address;

    public Human(String id,String name, String fname,String contact,String address)
    {
        Id = id;
        Name = name;
```

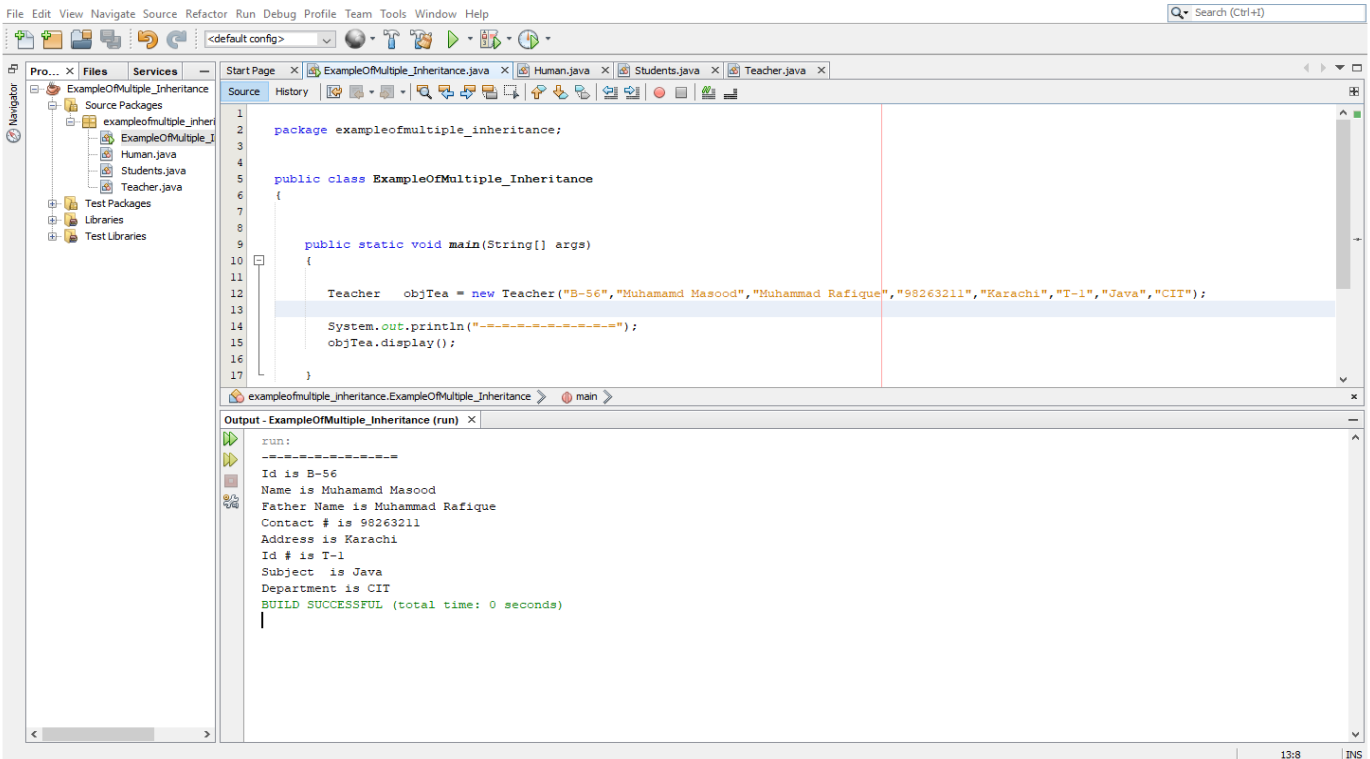
```
        FatherName = fname;
        Contact = contact;
        Address = address;
    }

     public void display()
    {
        System.out.println("Id is "+Id);
        System.out.println("Name is "+Name);
        System.out.println("Father Name is "+FatherName);
       System.out.println("Contact # is "+Contact);
       System.out.println("Address is "+Address);
    }
}
```

```
package exampleof. Hierarchical _inheritance;

public class Students extends Human {
  private String Roll;
  private int Attandance;
  private float Percentage;

   public Students(String i,String n,String f,String c,String a,String roll,int attandance,float per)
   {
      super(i,n,f,c,a);
      Roll =roll;
      Attandance = attandance;
      Percentage = per;
   }
    public void display()
   {
     super.display();
     System.out.println("Roll # is "+Roll);
     System.out.println("Attandance  is "+Attandance);
     System.out.println("Percentage is "+Percentage +" %");
     }
}
```

```
package exampleof. Hierarchical_inheritance;
public class Teacher extends Human {
  private String Id;
```

```
  private String Subject;
  private String Department;
   public Students(String i,String n,String f,String c,String a,String id,String subject,String department)
  {
     super(i,n,f,c,a);
     Id =id;
     Subject = subject;
     Department = department;
  }
   public void display()
  {
    super.display();
   System.out.println("Id # is "+Id);
   System.out.println("Subject  is "+Subject);
   System.out.println("Department is "+Department);
  }
}
```

```
package exampleof. Hierarchical _inheritance;
public class ExampleOfInheritance
{
  public static void main(String[] args)
  {
    Teacher  objTea = new Teacher("B-56","Muhamamd Masood","Muhammad Rafique","98263211","Karachi","T-1","Java","CIT");
    objTea.display();    }
}
```

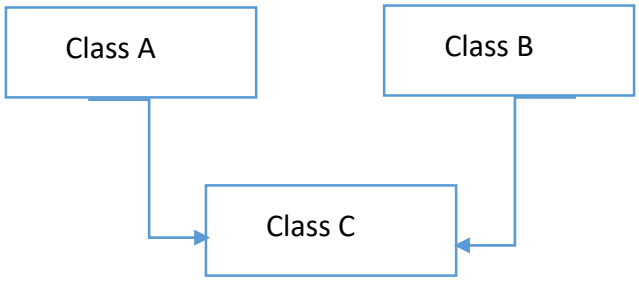


Multiple Inheritance

Multiple inheritances are a type of inheritance where a subclass can inherit features from more than one parent class. To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

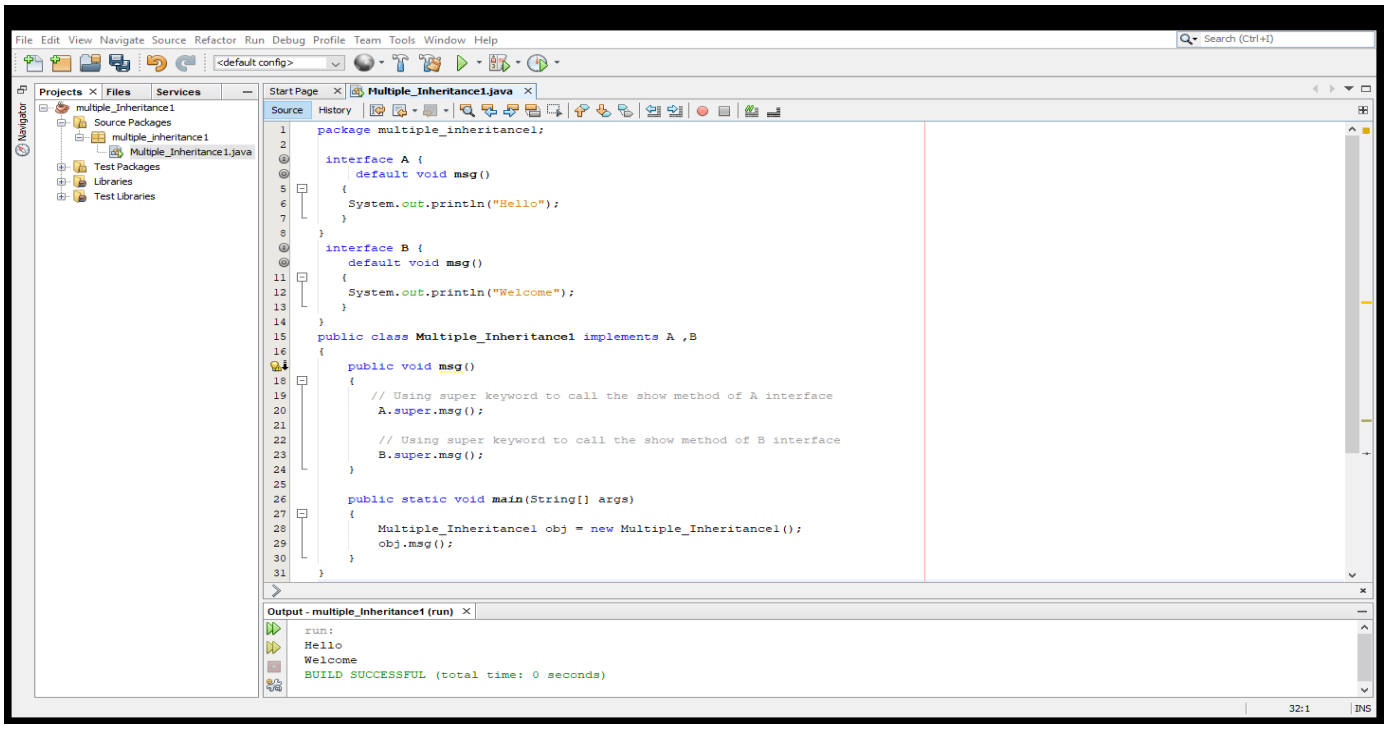
Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class. Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

As per above diagram, Class C extends Class A and Class B both.




```
class A
{
   void msg()
    {  System.out.println("Hello");
    }
}
class B
{
   void msg()
   { System.out.println("Welcome");
   }
}
class C extends A,B{  // suppose if it were  but there error because not support multiple inheritance

    public static void main(String args[])
  {
     C obj=new C();
    obj.msg();//Now which msg() method would be invoked?
  }
}
```

In java, we can achieve multiple inheritances only through Interfaces. In the image above, Class C is derived from interface A and B.

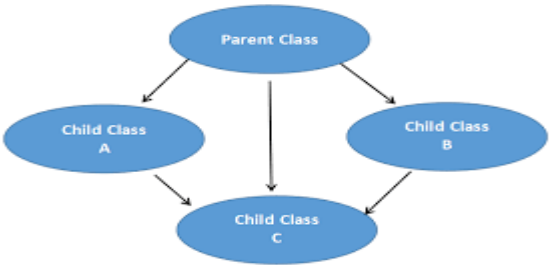



```
package multiple_inheritance1;


public class Multiple_Inheritance1 implements A ,B
{
   public void msg()
   {
     // Using super keyword to call the show method of A interface
      A.super.msg();

     // Using super keyword to call the show method of B interface
      B.super.msg();
   }

   public static void main(String[] args)
   {
     Multiple_Inheritance1 obj = new    Multiple_Inheritance1();
      obj.msg();
   }
}
```

```
interface A {
    default void msg()
  {
   System.out.println("Hello");
  }
}
 interface B {
   default void msg()
  {
   System.out.println("Welcome");
  }
}
```

Hybrid Inheritance

Hybrid inheritance is a type of inheritance that combines single inheritance and multiple inheritances. As multiple inheritances are not supported by Java, hybrid inheritance can also be achieved through interfaces only.



Interface

An interface is a fully abstract **class**.it is looks like a **class** but it is not a **class**. An interface can have methods and variables just like the **class** but the methods declared in interface are by default abstract (only method signature, without body), We use the ***interface*** keyword to create an interface in Java.

Why and when we use interface

We need as one of Java's core concepts, abstraction, polymorphism, and multiple inheritance are supported through this technology.

Syntax

```
class Demo implements MyInterface
{
  /* This class must have to implement both the abstract methods
   * else you will get compilation error
   */
  public void method1()
  {
            System.out.println("implementation of method1");
  }
  public void method2()
  {
     System.out.println("implementation of method1");
  }
}
```

```
interface MyInterface
{
  // All the methods are public abstract by default
  // As you see they have no body

  public void method1();
  public void method2();
}
```

```
package exampleofinterface;
public class ExampleOfInterface
{

   public static void main(String[] args)
   {
     TestInterface objInterface = new TestInterface();

     objInterface.method1();
     objInterface.method2();
   }
```

Example of Bank

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank
{
public float rateOfInterest()
    {
        return 9.15f;
    }
}
class PNB implements Bank
{
     public float rateOfInterest()
    {
        return 9.7f;
   }
}
class TestInterface2{
      public static void main(String[] args)
    {
      Bank b=new SBI();
      System.out.println("ROI: "+b.rateOfInterest());
    }
}
```

It is used to achieve total abstraction. Since java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances. It is also used to achieve loose coupling.

Exercise

Theory Questions

1. What is inheritance and its types.
2. Why and when Importance of Java inheritance
3. What do you mean by interface?
4. Why and when we use interface.
5. Why we use ***extends*** and ***final*** keyword in java language

Practical Questions

1. Write syntax of inheritance classes.
2. Write example of interface in the OOP.

Objective MCQ's

1) In java not supported to _____ inheritance.
   a) Multiple
   b) Multilevel
   c) single level
   d) not all
2) In Java, it is possible to _____ attributes and methods from one class to another
   a) constructor
   b) object
   c) inherit
   d) access modifier
3) Subclass is a class which inherits the other class. It is also called a _____ class.
   a) derived
   b) child
   c) super
   d) a and b both
4) If you don't want other classes to inherit from a class, use the_____ keyword:
   a) extends
   b) public
   c) final
   d) private
5) The _____ keyword is used to perform inheritance in Java
   a) final
   b) private
   c) public
   d) extends

6) interface is used to achieve _____.
   a) Total Abstract
   b) Multiple inheritance
   c) Loos coupling
   d) all