**Question 15. What Will Happen If A Class Extends Two Interfaces And They Both Have A Method With Same Name And Signature?**

**Answer :**

[Implementing two interfaces in a class with same method. Which interface method is overridden?](https://stackoverflow.com/questions/2801878/implementing-two-interfaces-in-a-class-with-same-method-which-interface-method)

As in interface, we are just declaring methods, concrete class which implements these both interfaces understands is that there is only one method(as you described both have same name and return type). so there should not be an issue with it. You will be able to define that method in concrete class.

But when two interface have a method with the same name but different return type then it will throw an error

Please look at below code:

public interface InterfaceA {

public void print();

}

public interface InterfaceB {

public int print();

}

public class ClassAB implements InterfaceA, InterfaceB {

public void print()

{

System.out.println("Inside InterfaceA");

}

public int print()

{

System.out.println("Inside InterfaceB");

return 5;

}

}

when compiler gets method "public void print()" it first looks in InterfaceA and it gets it.But still it gives compile time error that return type is not compatible with method of InterfaceB.

So it goes haywire for compiler.

In this way, you will not be able to implement two interface having a method of same name but different return type.

Well if they are both the same it doesn't matter. It implements both of them with a single concrete method per interface method.

#### Q9. In Java, Constructor over-riding is possible?

# **Answer:** **Why Constructors are not inherited in Java?**

Constructor is a block of code that allows you to create an object of class and has same name as class with no explicit return type.

Whenever a class (child class) extends another class (parent class), the sub class inherits state and behavior in the form of variables and methods from its super class but it does not inherit constructor of super class because of following reasons:

* Constructors are special and have same name as class name. So if constructors were inherited in child class then child class would contain a parent class constructor which is against the constraint that constructor should have same name as class name.

===================================================

#### Is it possible to override a private method in Java?

**Answer:**  
No, we cannot override a private method in Java as private methods scope is limited to that particular class only and they are not visible outside of that class, so they cannot be visible in derived class or subclass also. So the private methods are not overridden.

**Question 10. Can We Override A Private Method In Java?**

**Answer :**

No,  you cannot override a private method in Java because the private method is not inherited by the subclass in Java, which is essential for overriding. In fact, a private method is not visible to anyone outside the class and, more importantly, a call to the private method is resolved at compile time by using Type information as opposed to runtime by using the actual object.

=**Question 11. What Is Method Hiding In Java?**

**Answer :**

If a subclass has a static method with the same name and signature as a static method in the superclass, then the method in the super-class will be called irrespective of the fact whether it is called from child class or parent class.

## **What is Method Hiding?**

Method hiding is functionally very similar to methods overriding. In overriding if you create a method in sub-class with the same type and signature in sub-class then it allows calling of methods based on the type of instance.

In the case of static methods with the same type and signature in superclass and sub-class then, then the method in the subclass hides the method in the superclass.

## **Method Hiding Java Code**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | package com.test;    class Parent {        public static void first() {          System.out.println("Inside first method in parent class");      }        public void bar() {          System.out.println("Inside bar method in parent class");      }  }    class Child extends Parent {      // Hiding      public static void first() {          System.out.println("Inside first method in child class");      }        // Overriding      public void bar() {          System.out.println("Inside bar method in child class");      }  }    public class Code {        public static void main(String[] args) {          Parent p = new Parent();          Parent c = new Child();          System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Method Hiding\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");          p.first(); // This will call method in parent class          c.first(); // This will call method in parent class          System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Method overriding\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");          p.bar(); // This will call method in parent class          c.bar(); // This will call method in child class        }  } |

**Output:**



In the above example, the sub-class Child has static method foo () having the same name and signature as a static method in super-class Parent. When we call p.foo() and c.foo() it calls foo () method in parent class

unlike in method overriding where p.bar() is calling the method in the parent class and c.bar() calls the method in child class.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package com.test;    class Parent {        public static void foo() {          System.out.println("Inside foo method in parent class");      }        public void bar() {          System.out.println("Inside bar method in parent class");      }  }    class Child extends Parent {      // Hiding      public static void foo() {          System.out.println("Inside foo method in child class");      }        // Overriding      public void bar() {          System.out.println("Inside bar method in child class");      }  }    public class Code {        public static void main(String[] args) {          Parent p = new Parent();          Parent c = new Child();          System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Method Hiding\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");          p.foo(); // This will call method in parent class          c.foo(); // This will call method in parent class          System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Method overriding\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");          p.bar(); // This will call method in parent class          c.bar(); // This will call method in child class        }  } |

**Output:**



In the above example, the sub-class Child has static method foo () having the same name and signature as a static method in super-class Parent. When we call p.foo() and c.foo() it calls foo () method in parent class

unlike in method overriding where p.bar() is calling the method in the parent class and c.bar() calls the method in child class.

As static methods are resolved at compile time while compilation first parent class is complied and then child class, and we cannot have two static methods with same name both the foo methods are resolved as foo () method of the parent class.

**Summary**

If a subclass has a static method with the same name and signature as a static method in the superclass, then the method in the super-class will be called irrespective of the fact whether it is called from child class or parent class.

In case of method overriding we override method from parent class, i.e. if a subclass has a non-static method with the same name and signature as a non-static method in the superclass then respective methods are called depending upon reference used, i.e. if object of parent class is used to call non-static method in parent class then method from parent class is used and if object of child class is used to call non-static method in child class then method from child class is used.

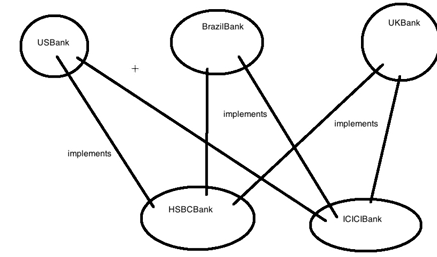
**Question 13. Can A Class Extends More Than One Class In Java?**

**Answer :**

No, a class can only extend just one more class in Java. Through Every class also, by default extend the java.lang.Object class in Java.

**USE INTERFACE CONCEPT AND MULTIPLE INHERITANCE**

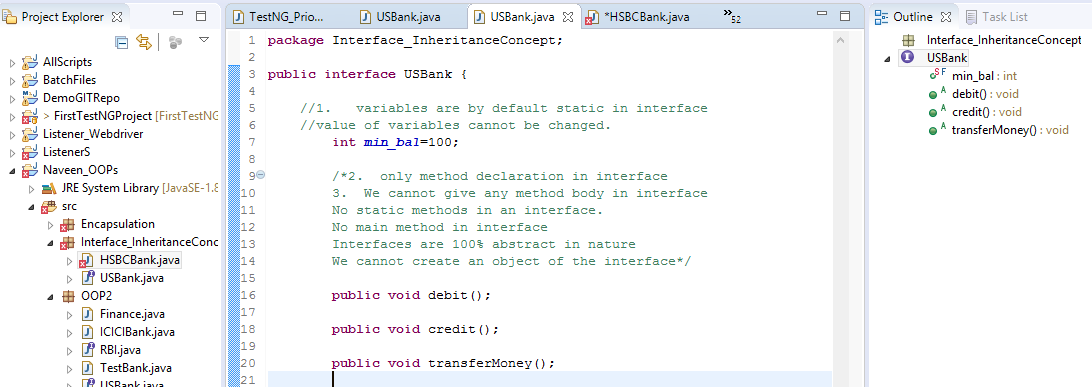
Create 3 interfaces; USBank, BrazilBank, and UKBank and create one class HSBC. Now HSBC wants to open a branch in US, Brazil, and UK so how they do that is . HSBC implements ; USBank, BrazilBank, and UKBank.



Now one more bank ICICIBank also wants to open their branches in US, Brazil, and UK. So if any bank has to open that particular bank class has to implement all the 3 interfaces

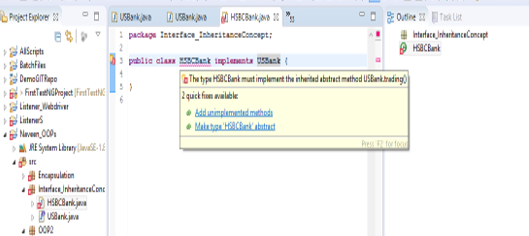
Lets now code the same

Create the USBank interface as below;

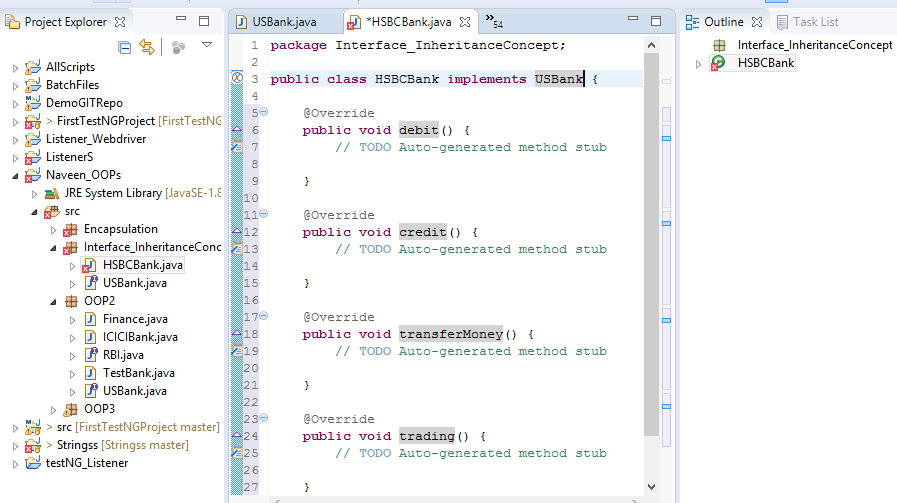


Create a HSBCBank class. Now if you want to create a relationship, you have to mention

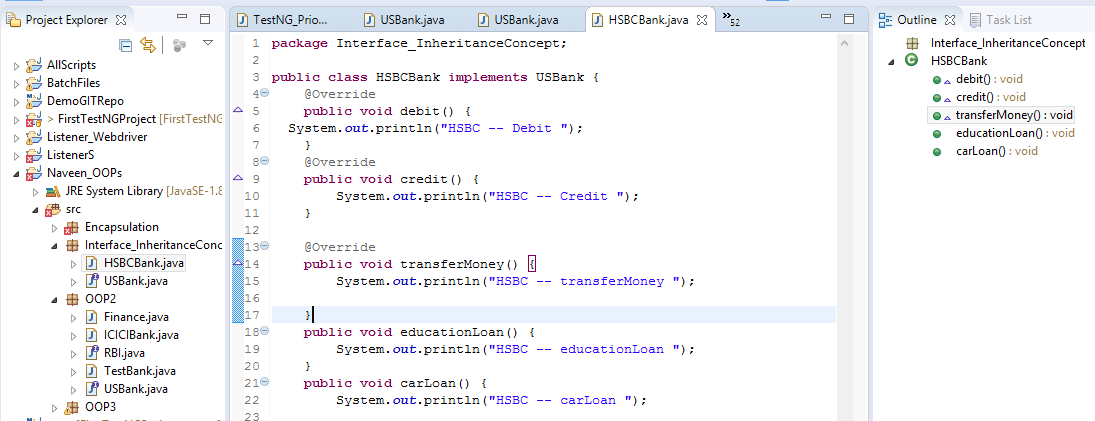
HSBCBank implements USBank



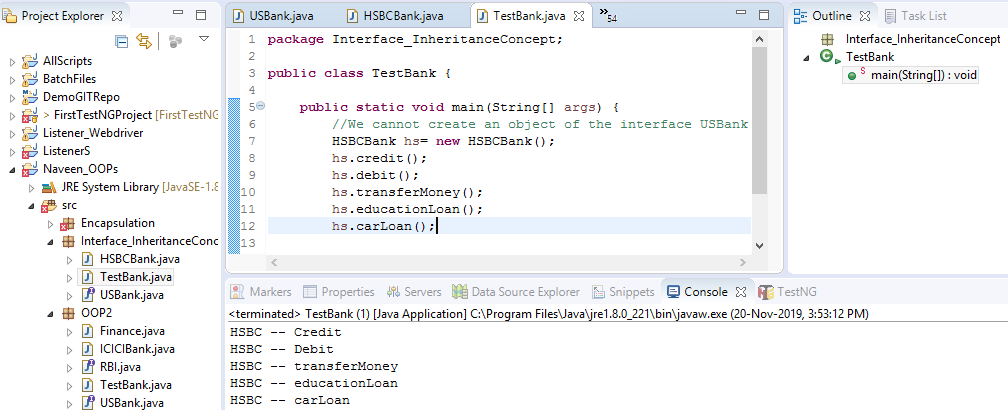
Now all the methods from USBank are implemented in HSBC bank



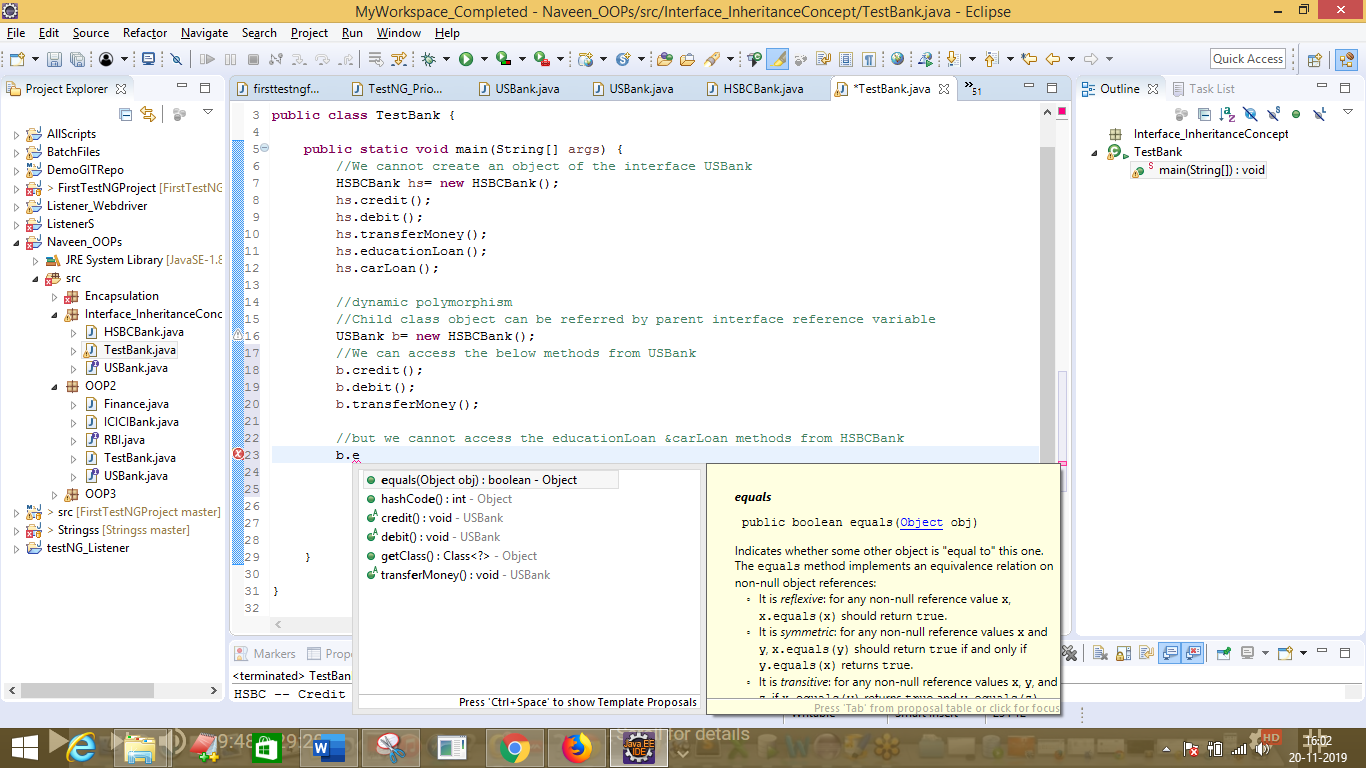
After implementing the above methods HSBCBank adds necessary logic changes according to them as below and also adds 2 of their own methods

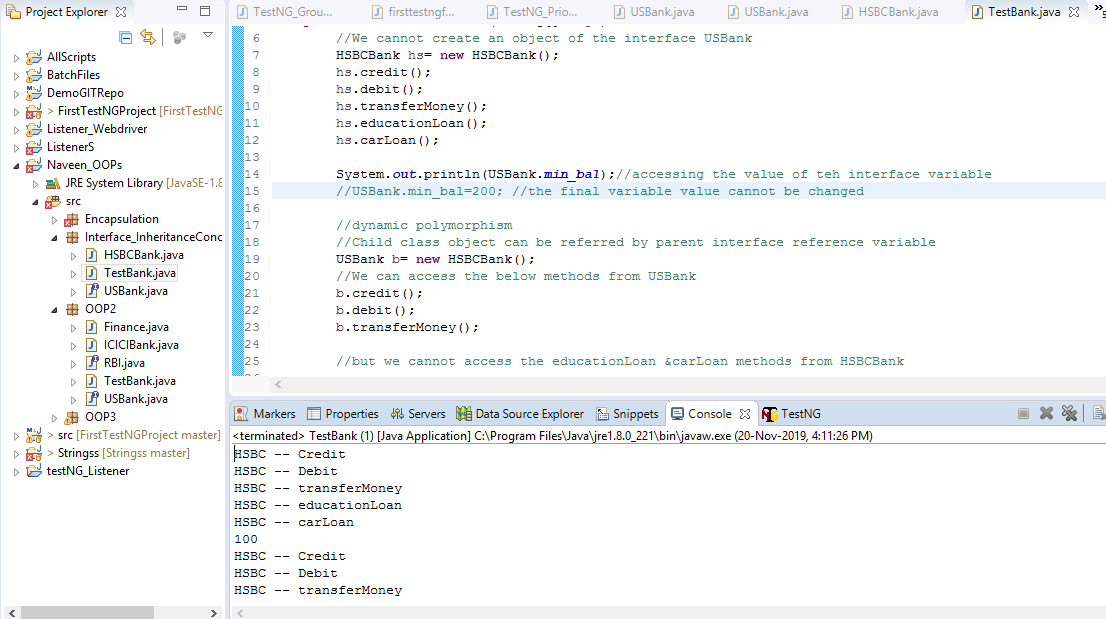


Now to check this or run this code, create a TestBank class as below and run the program



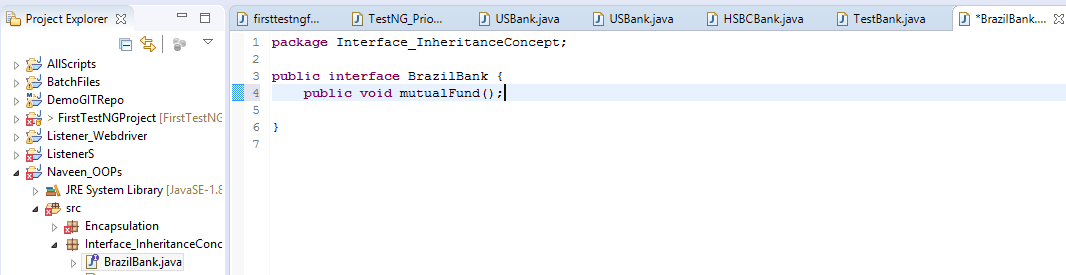
If a class is implementing any interface, it is mandatory to define/override all the methods of the interface.



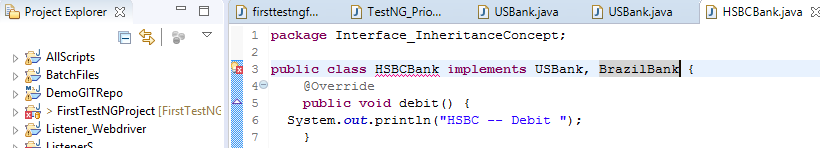


Interface variable value is final and static in nature

Now lets create another interface BrazilBank



Now if HSBCBank wants to implement BrazilBank then they have to use the implements keyword as below



You have to now also implement the method from BrazilBank into HSBCBank by clicking on Add Unimplemented methods

// we are achieving multiple inheritance

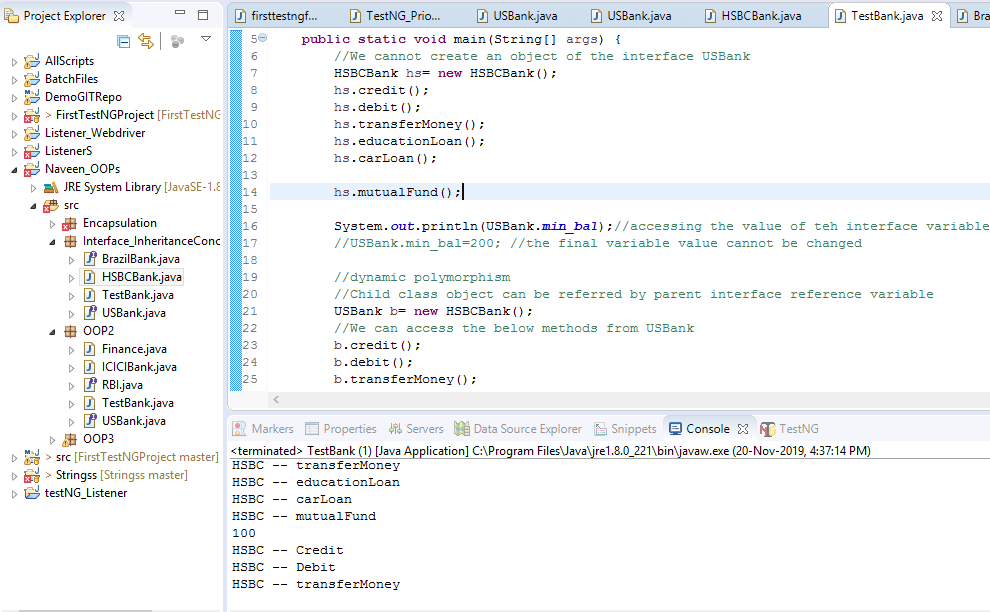
//Is a relationship

**public** **class** HSBCBank **implements** USBank, BrazilBank

IS A RELATIONSHIP means Interface to class relationship

HAS A RELATIONSHIP means class to class relationship.

Run the program



class Teacher {

String designation = "Teacher";

String collegeName = "Beginnersbook";

void does(){

System.out.println("Teaching");

}

}

public class PhysicsTeacher extends Teacher{

String mainSubject = "Physics";

public static void main(String args[]){

PhysicsTeacher obj = new PhysicsTeacher();

System.out.println(obj.collegeName);

System.out.println(obj.designation);

System.out.println(obj.mainSubject);

obj.does();

}

}

**An interface can also extend multiple interfaces.**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate multiple inheritance  // in interfaces  import java.io.\*;  interface intfA  {      void geekName();  }    interface intfB  {      void geekInstitute();  }    interface intfC extends intfA, intfB  {      void geekBranch();  }    // class implements both interfaces and provides  // implementation to the method.  class sample implements intfC  {      public void geekName()      {          System.out.println("Rohit");      }        public void geekInstitute()      {          System.out.println("JIIT");      }        public void geekBranch()      {          System.out.println("CSE");      }        public static void main (String[] args)      {          sample ob1 = new sample();            // calling the method implemented          // within the class.          ob1.geekName();          ob1.geekInstitute();          ob1.geekBranch();      }  } |

Output:

Rohit

JIIT

CSE

**Q7. Is it possible to override a private method in Java?**

**Answer:**  
No, we cannot override a private method in Java as private methods scope is limited to that particular class only and they are not visible outside of that class, so they cannot be visible in derived class or subclass also. So the private methods are not overridden.