Dt : 15/10/2022

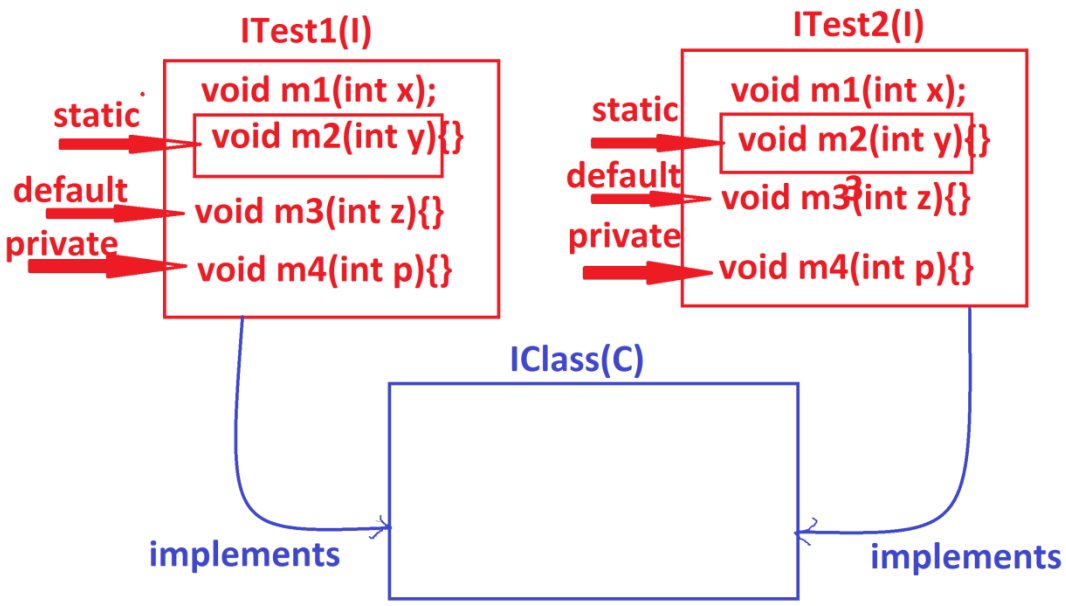
**imp*

Multiple Inheritance Models using Interfaces:

Model-1 : Extracting the features from more than one interface into a Class.

(Class implementing from more than one Interface)

Diagram:



Ex:

ITest1.java

```
package test;
public interface ITest1 {
  public abstract void m1(int x);
  public static void m2(int y) {
            System.out.println("====ITest1 static concrete m2(y)====");
            System.out.println("The value y:"+y);
  }
  default void m3(int z,int p) {
```

```
        System.out.println("===ITest1 default concrete m3(z)===");
        System.out.println("The value z:"+z);
        this.m4(p);
 }
 private void m4(int p) {
        System.out.println("====ITest private concrete m4(p)====");
        System.out.println("The value p:"+p);
 }
}
```

ITest2.java

```
package test;
public interface ITest2 {
        public abstract void m1(int x);
        public static void m2(int y) {
                System.out.println("====ITest2 static concrete m2(y)====");
                System.out.println("The value y:"+y);
         }
        default void m33(int z,int p) {
                System.out.println("===ITest2 default concrete m33(z)===");
                System.out.println("The value z:"+z);
                this.m4(p);
         }
        private void m4(int p) {
                System.out.println("====ITest private concrete m4(p)====");
                System.out.println("The value p:"+p);
         }
}
```

IClass.java

```
package test;
public class IClass implements ITest1,ITest2{
  public void m1(int x) {
        System.out.println("====abstract m1(x)====");
        System.out.println("The value x:"+x);
  }
}
```

DemoInterface5.java(MainClass)

```
package maccess;
```

```
import test.*;
public class DemoInterface5 {
        public static void main(String[] args) {
    IClass ob = new IClass();
    ob.m1(121);
    ITest1.m2(122);
    ITest2.m2(123);
    ob.m3(124,126);
    ob.m33(125,127);
        }
}
```

o/p:

====abstract m1(x)====

The value x:121

====ITest1 static concrete m2(y)====

The value y:122

====ITest2 static concrete m2(y)====

The value y:123

===ITest1 default concrete m3(z)===

The value z:124

====ITest private concrete m4(p)====

The value p:126

===ITest2 default concrete m33(z)===

The value z:125

====ITest private concrete m4(p)====

The value p:127

Note:

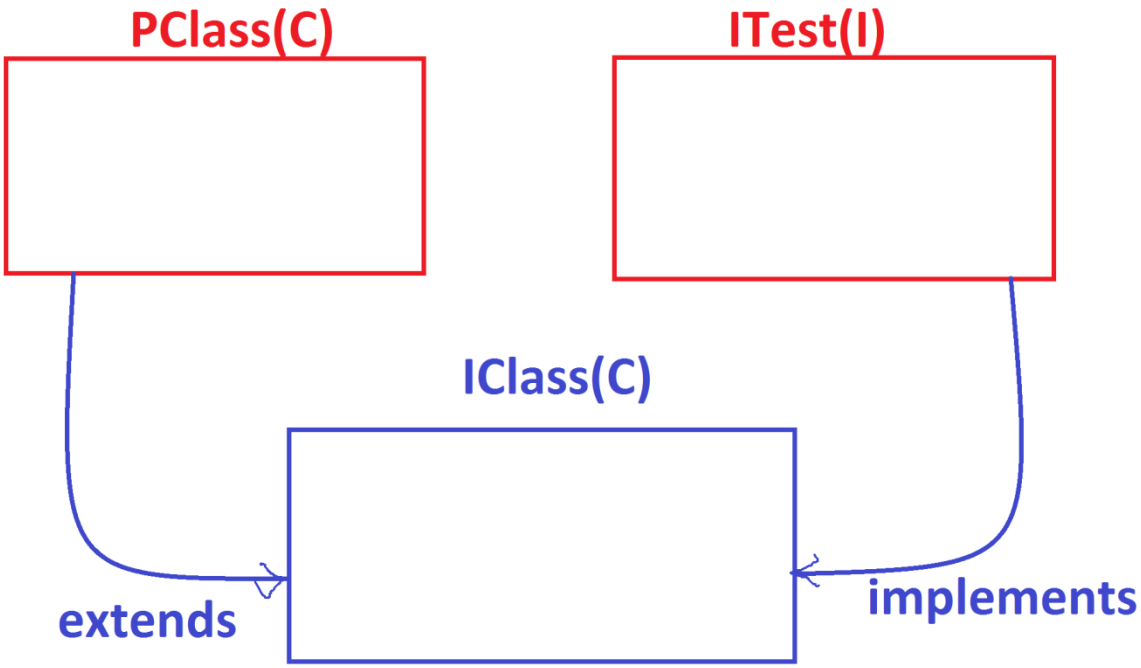
=>when we have same default concrete methods in Multiple-Inheritance

process then replication of programming components error is raise at compilation stage.

Model-2 : Extracting the features from one class and any number of Interfaces

(Class extending from one class and implementing from any number of Interfaces)

Diagram:



Ex:

PClass.java

```
package test;
public class PClass {
  public void m1(int x) {
            System.out.println("====PClass m1(x)====");
            System.out.println("The value x:"+x);
  }
}
```

ITest.java

```
package test;
public interface ITest {
   public abstract void m2(int y);
}
```

IClass.java

```
package test;
public class IClass extends PClass implements ITest{
    public void m2(int y) {
            System.out.println("====IClass m2(y)====");
            System.out.println("The value y:"+y);
    }
}
```

DemoInterface6.java(MainClass)

```
package maccess;
import test.*;
public class DemoInterface6 {
        public static void main(String[] args) {
      IClass ob = new IClass();
      ob.m1(11);
      ob.m2(12);
          }
}
```

o/p:

====PClass m1(x)====

The value x:11

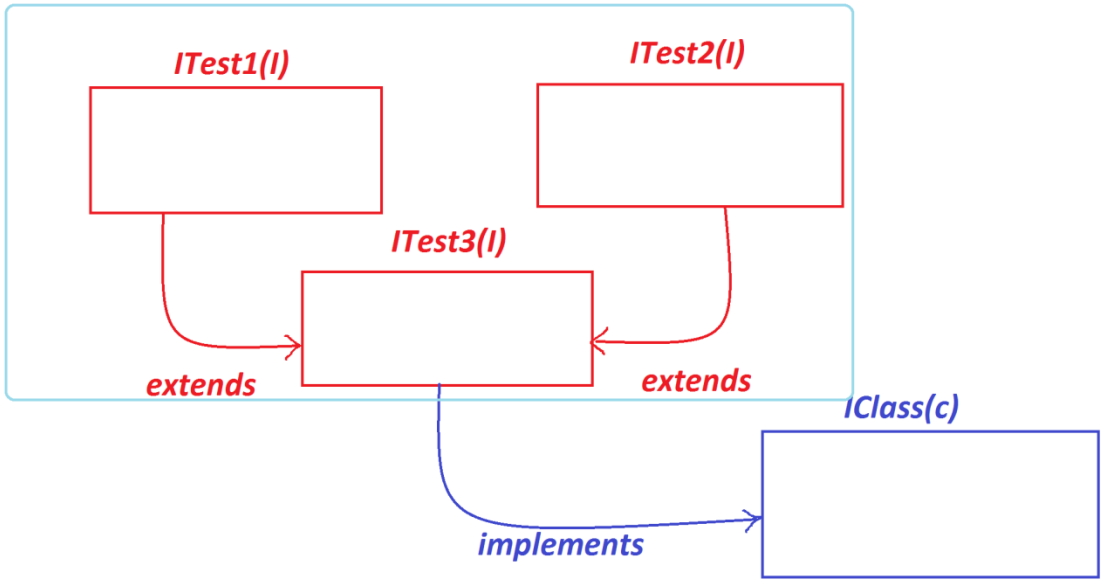
====IClass m2(y)====

The value y:12

Model-3 : Extracting the features from more than one interface into a Interface

(Interface extending from more than one Interface)

Diagram:



Ex:

ITest1.java

```
package test;
public interface ITest1 {
  public abstract void m1(int x);
}
```

ITest2.java

```
package test;
public interface ITest2 {
        public abstract void m2(int y);
}
```

ITest3.java

```
package test;
public interface ITest3 extends ITest1,ITest2{
        public abstract void m3(int z);
}
```

IClass.java

```
package test;
public class IClass implements ITest3{
   public void m1(int x) {
        System.out.println("x:"+x);
   }
   public void m2(int y) {
        System.out.println("y:"+y);
   }
   public void m3(int z) {
        System.out.println("z:"+z);
   }
}
```

DemoInheritance7.java(MainClass)

```
package maccess;
import test.*;
public class DemoInterface7 {
        public static void main(String[] args) {
     IClass ob = new IClass();
     ob.m1(11);
     ob.m2(12);
```

```
    ob.m3(13);
        }
}
```

o/p:

x:11

y:12

z:13

==========

Assignment:

Construct BankTransaction application using the following Layout:

step-1 : read pinNo

=>pinNo must be in 1111 or 2222 or 3333,else "Invalid pinNo".

=>If pinNo entered wrongly for three times then display the msg as "Transaction blocked".

step-2 : If the pinNo verified Successfully,then show the following choice:

1.WithDraw

2.Deposit

1.WithDraw:

=>Enter the amt

=>amt must be greater than Zero and multiples of 100,else "Invalid amt"

=>If the amt is validated Successfully,then create object for "WithDraw" class and pass amt as parameter to "process()" method.

=>Perform WithDraw_logic in process method,

=>If amt is less than balance then perform transaction,ele

display msg as "Insufficient fund"

o/p:

Amt withdrawn :

Balance amt :

Transaction Successfull

2.Deposit:

=>Enter the amt

=>amt must be greater than Zero and multiples of 100,else "Invalid amt"

=>If the amt is validated Successfully,then create object for "Deposit" class and pass amt as parameter to "process()" method.

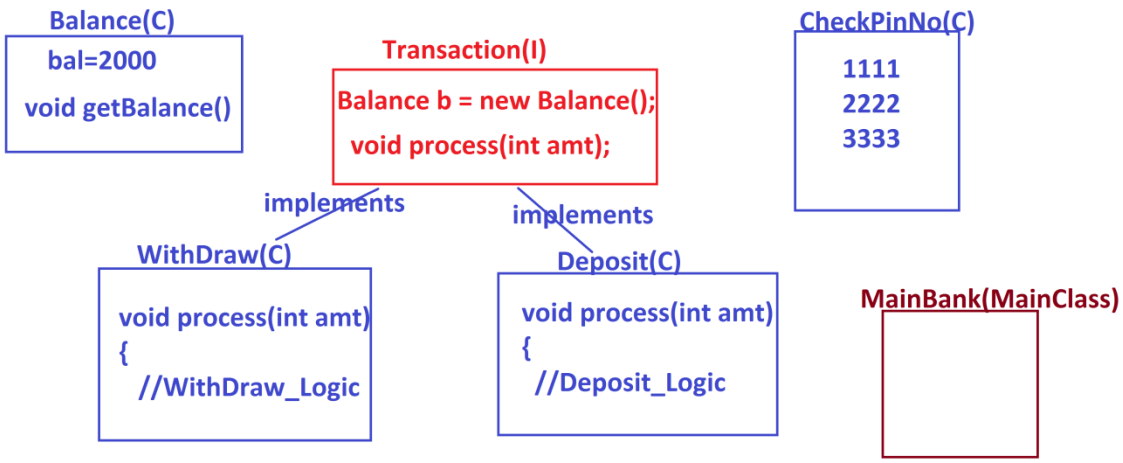
=>Perform Deposit_logic in process method,

o/p:

Amt Deposited :

Balance amt :

Transaction Successfull



==========

Dt : 17/10/2022

AbstractClasses in Java:

=>The classes which are declared with abstract keyword are known as AbstractClasses.

=>AbstractClasses will hold Variables,abstract methods,Concrete methods blocks,Constructors and features.

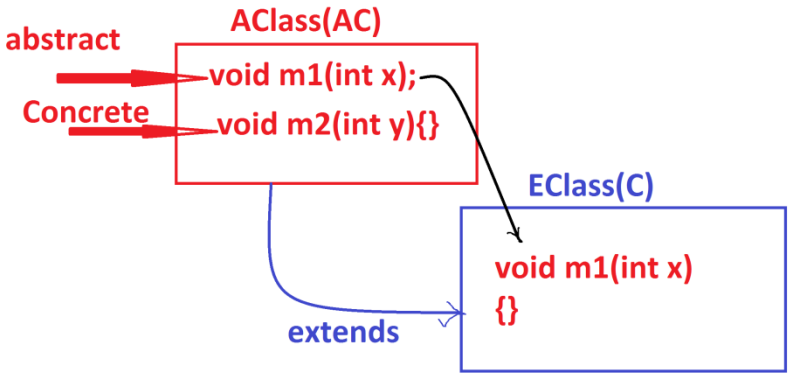
=>AbstractClasses cannot be instantiated,which means we cannot create object for abstract classes.

=>we must use "abstract" keyword to declare abstract methods in abstract classes.

=>These abstract classes are extended to classes and the classes are known as "extention classes" or "implemented classes".

=>These "extention classes" must construct body for abstract methods of abstract classes

Diagram:



EClass ob = new EClass();

Ex:

AClass.java

```
package test;
public abstract class AClass {
  public abstract void m1(int x);
  public void m2(int y) {
        System.out.println("====Concrete m2(y)====");
        System.out.println("The value y:"+y);
  }
}
```

EClass.java

```
package test;
public class EClass extends AClass{
  public void m1(int x) {
        System.out.println("====abstract m1(x)====");
        System.out.println("The value x:"+x);
  }
}
```

DemoAbstractClass.java(MainClass)

```
package maccess;
```

```
import test.*;
public class DemoAbstractClass {
        public static void main(String[] args) {
    //AClass ob = new AClass();//Error
                EClass ob = new EClass();
                ob.m1(12);
                ob.m2(13);
        }
}
```

o/p:

====abstract m1(x)====

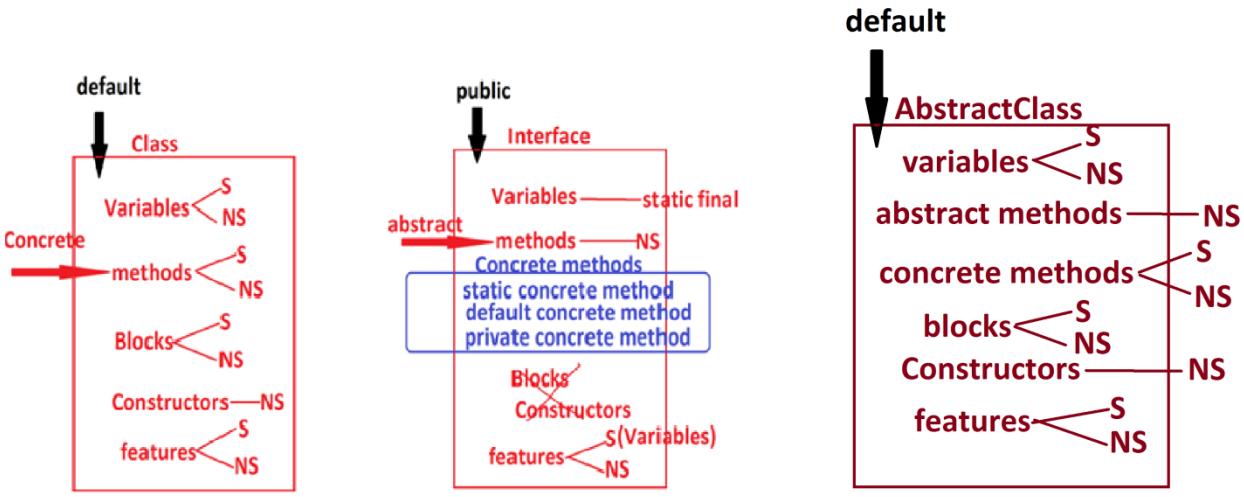
The value x:12

====Concrete m2(y)====

The value y:13

==========

Comparision Diagram:



==========

faq:

wt is the diff b/w

(i)Class

(ii)Interface

=>Class can be instantiated,but Interface cannot be Instantiated.

=>Class will hold only concrete methods,but Interface can hold both abstract methods and Concrete methods.

=>Programming components in classes are automatically "default",but in interfaces automatically "public".

=>Variables in classes are user-choice,but variables in interfaces are automatically "static final"

=>Classes can hold "blocks and Constructors",but Interfaces cannot hold "blocks and Constructors".

==========

faq:

wt is the diff b/w

(i)Class

(ii)Abstract Class

=>Class can be instantiated,but abstract class cannot be instantiated.

=>Class will hold holy Concrete methods,but abstract classes can hold both "abstract methods and Concrete methods.

==========

faq:

wt is the diff b/w

(i)Interface

(ii)AbstractClass

=>The programming components in interface are automatically "public",but programming components in abstract classes are automatically "default"

=>Variables in Interfaces are automatically "static and final",but variables in abstract classes are user-choice.

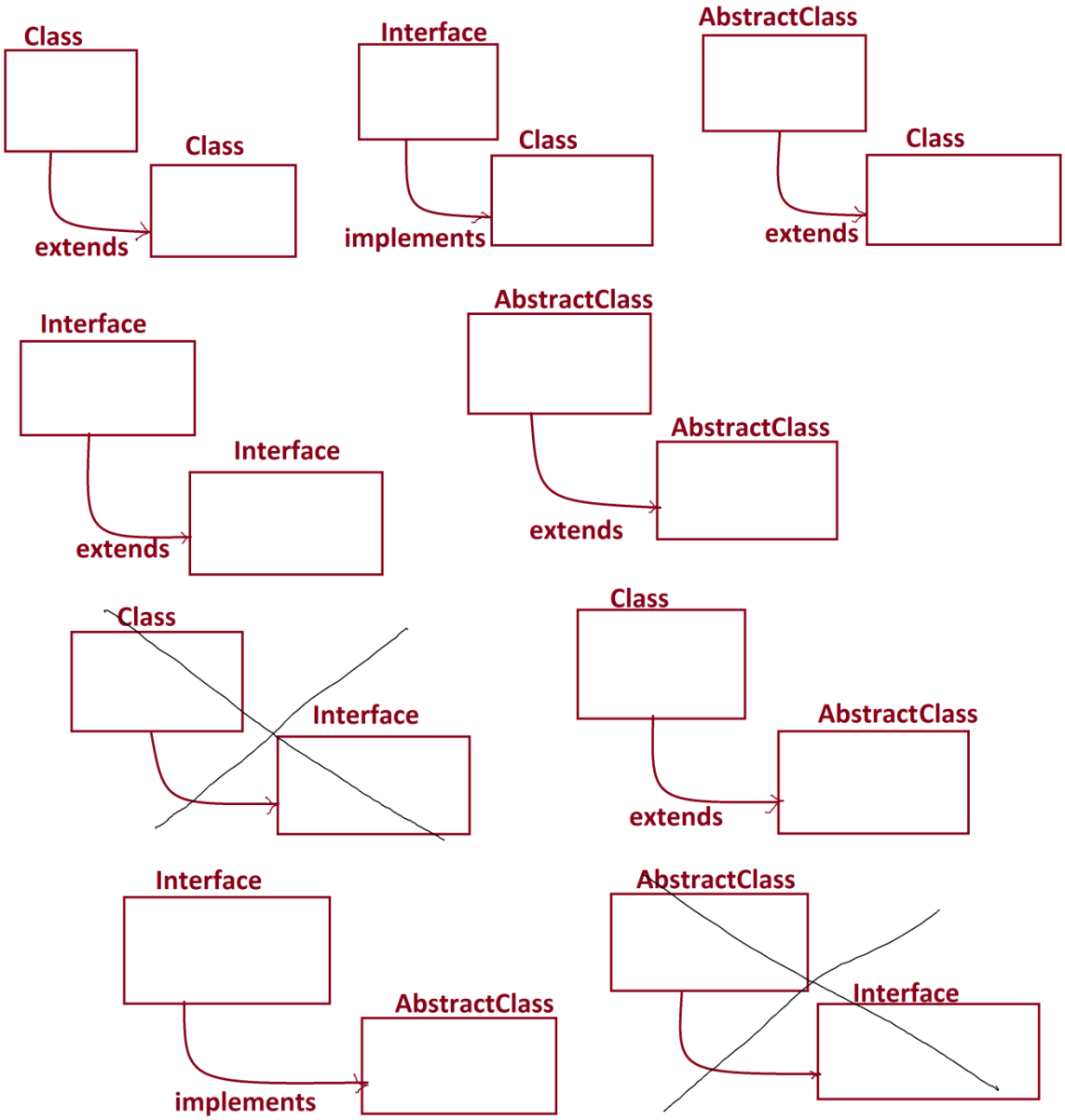
=>Interfaces cannot hold "blocks and Constructors",but abstract classes can hold "blocks and Constructors".

==========

****imp***

Single Inheritance Models:

Diagrams:


Class
Class
extends
Interface
Class
implements
AbstractClass
Class
extends
Interface
Interface
extends
AbstractClass
AbstractClass
extends
Class
Interface
Class
AbstractClass
extends
Interface
AbstractClass
implements
AbstractClass
Interface

==========