Dt : 22/8/2023

Note:

=>In realtime application development,the inheritances are categorized into two types:

1.Single Inheritance

2.Multiple Inheritance

1.Single Inheritance:

=>The process of taking the features(components) from one class at-a-time is known as Single Inheritance.

Ex:

above programs

2.Multiple Inheritance:

=>The process of taking the features(components) from more than one class at-a-time is known as Multiple Inheritance.

Diagram:

==

Note:

=>Multiple Inheritance Process using classes not available in Java, because it generate replication(duplicate) of programming components and

raises ambiguity.The ambiguity state applications will give wrong results.

=>Multiple Inheritance process in Java can be performed using Interfces.

===

****imp***

Interfaces in Java:

=>Interface is a collection of Variables,abstract methods and concrete methods from Java8 version onwards.

(Upto Java7 version,Interface is a Collection of variables and abstract methods,but cannot hold Concrete methods)

faq:

define abstract methods?

=>The methods which are declared without method_body are known as abstract methods.

Structure of abstract methods:

return_type method_name(para_list);

faq:

define Concrete methods?

=>The methods which are declared with method_body are known as Concrete methods.

Structure of Concrete methods:

```
return_type method_name(para_list)
{
 //method_body
}
```

Coding rules of Interface:

Rule-1 : we use "interface" keyword to construct interfaces

syntax:

```
interface Interface_name
{
  //Interface_body
}
```

Rule-2 : The programming components which are declared within the interface are automatically "public"

Note:

=>The programming components which are declared in classes without any access modifiers are considered as "default"

Rule-3 : The interfaces can be declared with both primitive datatype and NonPrimitive datatype variables

Rule-4 : The variables which are declared within the interface are automatically static and final variables

Note:

(i)static variables in interfaces will get the memory within the interface while interface loading and can be accessed with interface_name

(ii)final variables must be initialized with values and once initialized cannot be modified.

(final variables are also known as Constant Variables or Secured Variables)

Rule-5 : The methods which are declared within the interface are automatically NonStatic abstract methods.

(static abstract methods are not available)

Rule-6 : Interfaces cannot be instantiated in Java,which means we cannot create object for Interfaces.

Rule-7 : Interfaces are implemented to classes using "implements" keyword

and the classes are known as implementation classes.

Rule-8 : These implementation classes must construct the body for all abstract methods of Interface.

ProjectName : Interface_App1

packages,

p1 : ITest.java

```java
package p1;
public interface ITest
{
     public static final int k=100;
     public abstract void dis();

}
```

p1 : IClass.java

```java
package p1;
public class IClass implements ITest{
   public void dis() {
        System.out.println("====Implemented-
dis()=====");
        System.out.println("The value k:"+k);
   }
}
```

p2 : DemoInterface1.java(MainClass)

```java
package p2;
import p1.*;
public class DemoInterface1 {
	public static void main(String[] args) {
	   //ITest ob = new ITest();//Error
		IClass ob = new IClass();//Implemented Object
		ob.dis();
	}
}
```

o/p:

```
====Implemented-dis()=====
The value k:100
```

diagram:

===

Dt : 23/8/2023

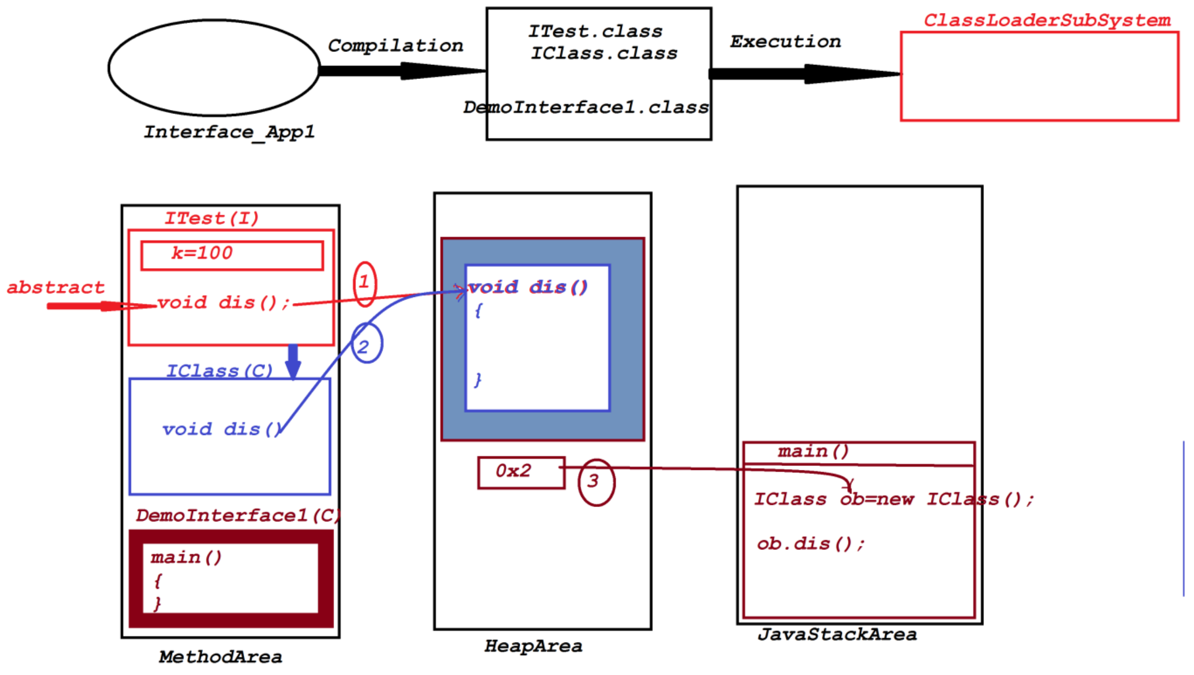
Execution flow of above program:

ClassFiles:

ITest.class

IClass.class

DemoInterface1.class(MainClass)



===

Rule-9 : Implementation classes can also be declared with NonImplemented methods

ProjectName : Interface_App2

packages,

p1 : ITest.java

```
package p1;
public interface ITest {
   public abstract void m1(int x);
```

```
    public abstract void m2(int y);
}
```

p1 : IClass.java

```
package p1;
public class IClass implements ITest{
     public void m1(int x)//Implemented and Overriding 
methods
      {
      System.out.println("===Implemented-m1(x)====");
      System.out.println("The value x:"+x);
      }
     public void m2(int y)//Implemented and Overriding 
methods
      {
      System.out.println("===Implemented-m2(y)====");
      System.out.println("The value y:"+y);
      }
     public void m3(int z)//NonImplemented method
      {
      System.out.println("===NonImplemented-m3(z)====");
      System.out.println("The value z:"+z);
      }
}
```

p2 : DemoInterface2.java(MainClass)

```
package p2;
import p1.*;
public class DemoInterface2 {
      public static void main(String[] args) {
         IClass ob = new IClass();//Implementation Object
         ob.m1(11);
         ob.m2(12);
         ob.m3(13);
      }
}
```

o/p:

```
===Implemented-m1(x)====
The value x:11
===Implemented-m2(y)====
The value y:12
===NonImplemented-m3(z)====
The value z:13
```

===
=

faq:

wt is the diff b/w

(i)Implemented methods

(ii)NonImplemented methods

(i)Implemented methods:

=>The methods which are taken from the interface and constructed body part of implementation classes are known as Implemented methods.

(ii)NonImplemented methods:

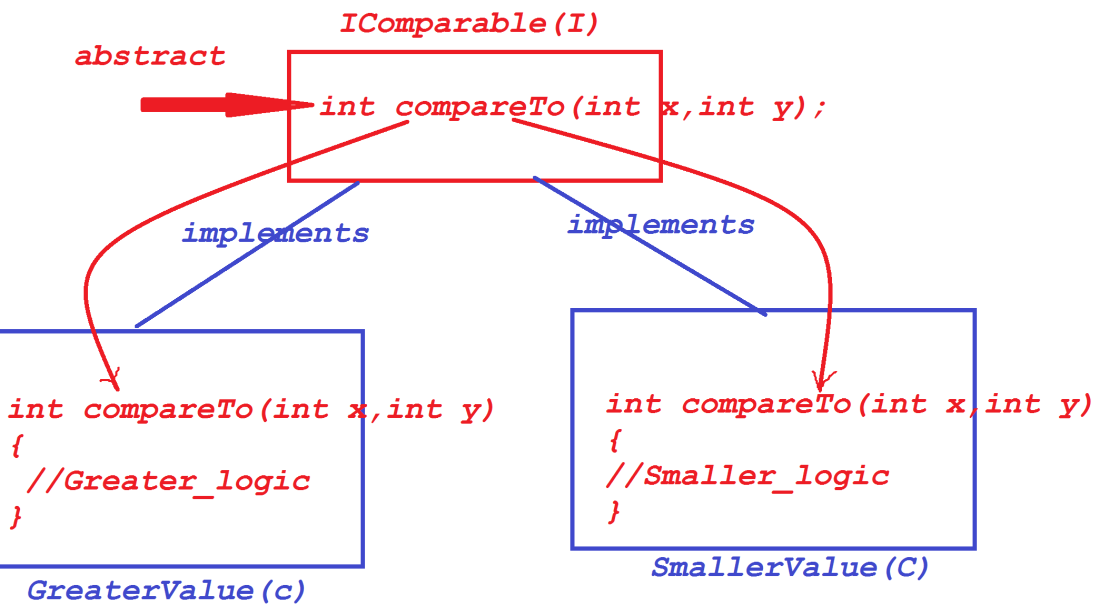
=>The methods which are constructed directly part of implementation classes are known as NonImplemented methods,which means the methods which

are not taken from the Interface.

==

Rule-10 : The interfaces can be implemented to any number implementation classes.

Layout:



ProjectName : Interface_App3

packages,

p1 : IComparable.java

```
package p1;
public interface IComparable {
   public abstract int compareTo(int x,int y);
}
```

p1 : GreaterValue.java

```
package p1;
public class GreaterValue implements IComparable{
   public int compareTo(int x,int y) {
       if(x>y) return x;
       else return y;
   }
}
```

p1 : SmallerValue.java

```
package p1;
public class SmallerValue implements IComparable{
   public int compareTo(int x,int y) {
        if(x<y) return x;
        else return y;
   }
}
```

p2 : DemoInterface3.java(MainClass)

```
package p2;

import java.util.*;

import p1.*;

public class DemoInterface3 {

        public static void main(String[] args) {

    Scanner s = new Scanner(System.in);

    System.out.println("Enter the value-1:");

    int v1 = s.nextInt();
```

```
System.out.println("Enter the value-2:");

int v2 = s.nextInt();

if(v1>0 && v2>0)

{

    System.out.println("****Choice****");

    System.out.println("\t1.GreaterValue"

                    + "\n\t2.SmallerValue");

    System.out.println("Enter the Choice:");

    int choice = s.nextInt();

    switch(choice)

    {

    case 1:

          GreaterValue gv = new GreaterValue();

          int res1 = gv.compareTo(v1, v2);

          System.out.println("GreaterValue:"+res1);

          break;

    case 2:

          SmallerValue sv = new SmallerValue();

          int res2 = sv.compareTo(v1, v2);

          System.out.println("SmallerValue:"+res2);

          break;

    default:
```

```
                    System.out.println("Invalid Input...");
               }//end of switch
          }//end of if
          else
          {
               System.out.println("Invalid input..");
          }
          s.close();
               }
}
```

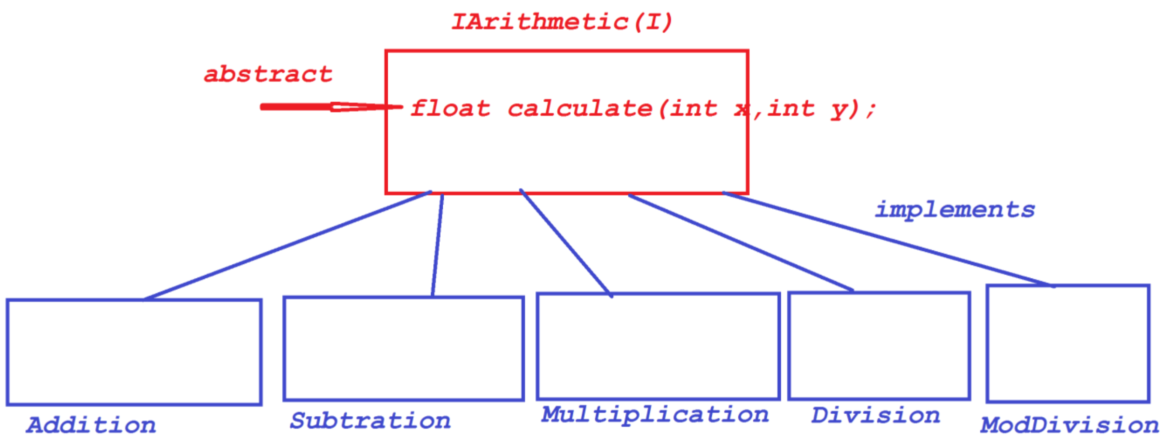
o/p:

```
Enter the value-1:
12
Enter the value-2:
13
****Choice****
        1.GreaterValue
        2.SmallerValue
Enter the Choice:
1
GreaterValue:13
```

==

Assignment:

Construct IArithmetic-Application using following Layout:



==

Rule-11 : Interfaces can be declared with Concrete methods.

==

****imp***

Concrete Methods in Interfaces:(Java8 - new feature)

=>Java8 version onwards the interfaces can be declared with concrete methods.

=>The following concrete methods can be declared in Interfaces:

(a)static concrete methods(Java8 - 2014)

(b)default concrete methods(Java8 - 2014)

(c)private concrete methods(Java9 - 2017)

===