Dt : 6/9/2023

**imp*

InnerInterfaces in Java:

(i)InnerInterfaces in Classes:

=>we can also declare InnerInterfaces in Classes and which can be Static member InnerInterfaces or NonStatic member InnerInterfaces.

(ii)InnerInterfaces in Interfaces:(Nested Interfaces)

=>we can also declare InnerInterfaces in Interfaces and which are automatically static member InnerInterfaces.

(iii)InnerInterfaces in AbstractClasses:

=>we can also declare InnerInterfaces in AbstractClasses and which can be static member InnerInterfaces or NonStatic member InnerInterfaces

Ex:

ProjectName : App_InnerInterfaces

packages,

p1 : SubClass.java

```
package p1;
public class SubClass
{
    public interface ITest1
    {
```

```
        public abstract void m1(int a);
    }//Instance member InnerInterface
    public static interface ITest11
    {
        public abstract void m11(int b);
    }//Static member InnerInterface
}//OuterClass
```

p1 : ITest.java

```
package p1;
public interface ITest
{
    public static interface ITest2
    {
        public abstract void m2(int c);
    }//Static member InnerInterface
}//OuterInterface
```

p1 : AClass.java

```
package p1;
public abstract class AClass
{
     public interface ITest3
     {
     public abstract void m3(int d);
     }//Instance member InnerInterface
     public static interface ITest33
     {
     public abstract void m33(int e);
     }//Static member InnerInterface
}//OuterAbstractClass
```

p2 : DemoInnerInterfaces.java(MainClass)

```
package p2;
import p1.*;
```

```
public class DemoInnerInterfaces
{
     public static void main(String[] args)
     {
        System.out.println("****InnerInterfac in
Class*****");
        SubClass.ITest1 ob1 = (int a)->
        {
       System.out.println("---m1(a)---");
       System.out.println("The value a:"+a);
        };
        SubClass.ITest11 ob11 = (int b)->
        {
        System.out.println("---m11(b)---");
            System.out.println("The value b:"+b);
        };
        ob1.m1(11);
        ob11.m11(12);
        System.out.println("****InnerInterface in
Interface****");
        ITest.ITest2 ob2 = (int c)->
        {
        System.out.println("---m2(c)---");
             System.out.println("The value c:"+c);
        };
        ob2.m2(13);
        System.out.println("****InnerInterfaces in
AbstractClasses****");
        AClass.ITest3 ob3 = (int d)->
        {
        System.out.println("---m3(d)---");
             System.out.println("The value d:"+d);
        };
        AClass.ITest33 ob33 = (int e)->
        {
        System.out.println("---m33(e)---");
             System.out.println("The value e:"+e);
        };
        ob3.m3(15);
        ob33.m33(16);
```

```
        }
}
```

o/p:

```
****InnerInterfac in Class*****
---m1(a)---
The value a:11
---m11(b)---
The value b:12
****InnerInterface in Interface****
---m2(c)---
The value c:13
****InnerInterfaces in AbstractClasses****
---m3(d)---
The value d:15
---m33(e)---
The value e:16
============================================================
```

**imp*

InnerAbstractClasses in Java:

(i)InnerAbstractClasses in Classes:

=>we can also declare InnerAbstractClasses in Classes and which

can be static member InnerAbstractClasses or NonStatic member InnerAbstractClasses.

(ii)InnerAbstractClasses in Interfaces:

=>We can also declare InnerAbstractClasses in Interfaces and which are automatically Static member InnerAbstractClasses.

(iii)InnerAbstractClasses in AbstractClasses:(Nested AbstractClasses)

=>we can also declare InnerAbstractClasses in AbstractClasses and which can be static member InnerAbstractClasses or NonStatic member InnerAbstractClasses.

Ex:(Assognment)

==========

faq:

define Encapsulation process?

=>The process of binding all the programming components into a single unit class is known as "Encapsulation process".

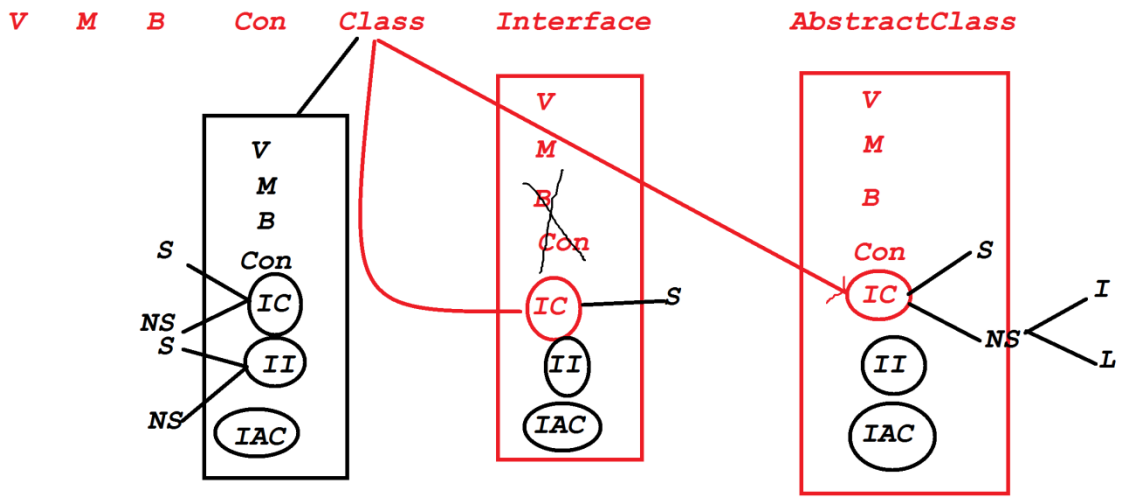
=>Class can hold Variables,Concrete methods,Blocks,Constructors, InnerClasses,InnerInterafces and InnerAbstractClasses.

=>Interface can hold Variables,Abstract methods,Concrete methods, InnerClasses,InnerInterfaces and InnerAbstractClasses

=>AbstractClass can hold variables,Abstract methods,Concrete methods,

Blocks,Constructors,InnerClasses,InnerInterfaces and InnerAbstract Classes.

Diagram:



==========

==

Summary of Programming Components:(Java Alphabets)

(a)variables

1.Primitive datatype variables(Values)

(i)Static Variables

(ii)NonStatic Variables

=>Instance Variables

=>Local Variables

2.NonPrimitive datatype variables(Object references)

(i)Static Variables

(ii)NonStatic Variables

=>Instance Variables

=>Local Variables

(b)Methods

1.Static methods

2.NonStatic methods(Instance methods)

(c)Blocks

1.static blocks

2.NonStatic blocks(Instance blocks)

(d)Constructors

=>NonStatic Constructors

(There is no Concept of Static Constructors in Java)

(e)Classes

1.static classes(Only as InnerClasses)

2.NonStatic Classes

(f)Interfaces

1.static Interfaces(Only as InnerInterfaces)

2.NonStatic Interfaces

(g)AbstractClasses

1.static AbstractClasses(Only as InnerAbstractClasses)

2.NonStatic AbstractClasses

==========

**imp*

Exception Handling Process:

define Exception?

=>The disturbance which is occured from the application is known as Exception.

define Exception handling process?

=>The process which is used to handle the exception is known as Exception handling process.

=>The use the following blocks in Exception Handling process:

1.try

2.catch

3.finally

=>These blocks are executed automatically when exception is raised inthe application

1.try:

=>try-block will hold the statements which are going to raise the exception.

=>when exception is raised in try-block,then one object is created for exception-type-class and the object reference is thrown onto catch block

syntax:

```
try
{
 //statements;
}
```

2.catch:

=>catch-block will hold the reference thrown from the try-block.

=>The required msgs are generated from catch-block.

syntax:

```
catch(exception-type-class ref-var)
{
 //msg
```

}

3.finally:

=>finally block is part of exception handling process,but executed independently without depending on the exception.

=>In realtime finally-block will hold resource closing operations like IO Close,File Close,DB Close,...

syntax:

```
finally
{
 //statements;
}
```

==========

*

imp

define "Throwable"?

=>"Throwable" is a class from java.lang package and which is root of Exception-handling-process.

=>This "Throwable" class is extended into the following two SubClasses:

1.Error class

2.Exception class

Hierarchy of "Throwable":