***Dt : 22/10/2022***

***Advantage of LambdaExpression:***

***=>when we use LambdaExpressions separate class files are not generated,***

***in this process the loading time of execution process is saved and generate***

***HighPerformance of an application.***

***Coding rules of LambdaExpression:***

***Rule-1 : The targer interface of LambdaExpression must be "Functional***

***Interface",which means the interface must be declared with only***

***one abstract method,but can be declared with any number of concrete***

***methods.***

***Rule-2 : The parameters which are declared in LambdaExpressions,the same***

***parameter names must not be taken as Local Variables in the same***

***method\_scope.***

***Rule-3 : LambdaExpressions must access the variables from Functional***

***interfaces using Interface\_name***

***==============================================================***

***faq:***

***wt is the diff b/w***

***(i)Normal Interface***

***(ii)Functional Interface***

***(iii)Marker Interface***

***(i)Normal Interface:***

***=>The Interfaces which can be declared with any number of abstract***

***methods are known as Normal Interfaces.***

***Ex:***

***java.util.Collection<E>***

***java.util.Map<K,V>***

***(ii)Functional Interface:***

***=>The interfaces which are declared with only one abstract method are***

***known as Functional Interfaces.***

***Ex:***

***java.lang.Runnable***

***java.lang.Comparable***

***(iii)Marker Interface:***

***=>The empty Interfaces are known as Marker Interfaces or Tagging***

***Interfaces.***

***Ex:***

***java.lang.Cloneable***

***java.io.Serializable***

***===================================================================***

***Note:***

***=>LambdaExpressions are used for Functional Interfaces and Anonymous***

***InnerClasses are used for Normal Interfaces.***

***================================================================***

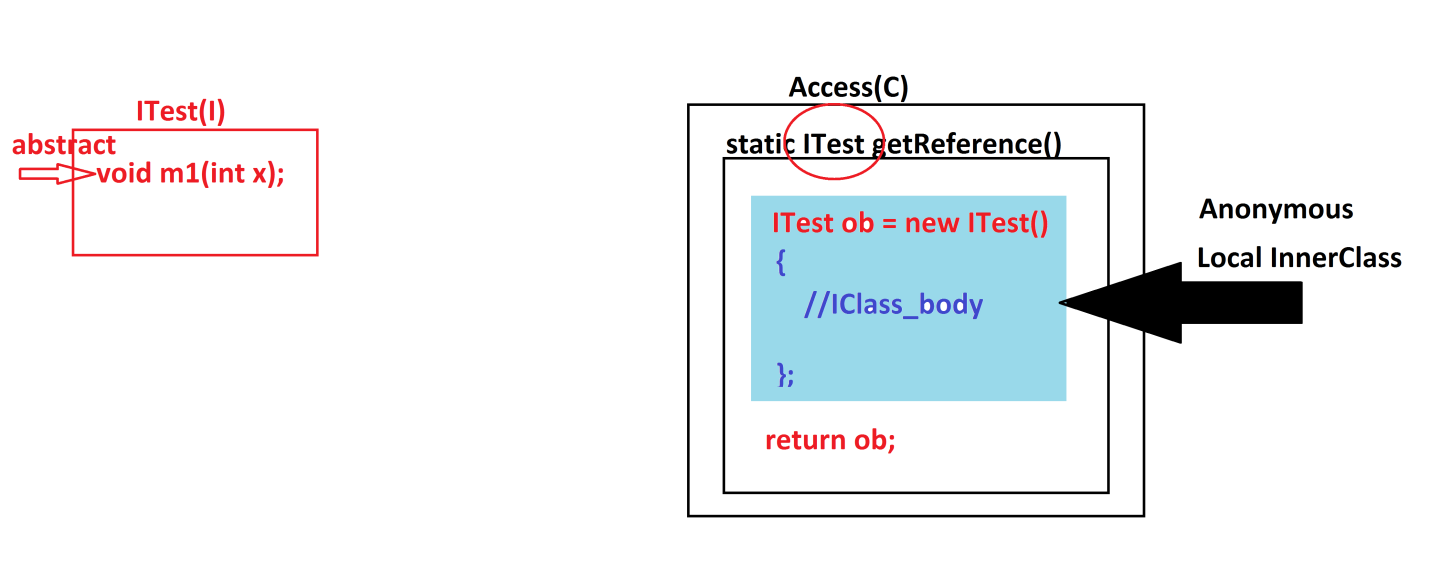
***\*imp***

***Design Models in Applications:***

***Model-1 : Generating implementation object for Interface by declaring***

***implementation class as "Anonymous Local InnerClass"***

***Diagram:***

******

***Ex:***

***ITest.java***

***package test;***

***public interface ITest {***

***public abstract void m1(int x);***

***}***

***Access.java***

***package test;***

***public class Access {***

***public static ITest getReference()***

***{***

***ITest ob = new ITest()***

***{***

***public void m1(int x)***

***{***

***System.out.println("===method m1(x)====");***

***System.out.println("The value x:"+x);***

***}***

***};***

***return ob;***

***}//OuterClass method***

***}//OuterClass***

***DemoModel1.java(MainClass)***

***package maccess;***

***import test.\*;***

***public class DemoModel1 {***

***public static void main(String[] args) {***

***ITest ob = Access.getReference();***

***//This method will create implementation object of ITest***

***ob.m1(12);***

***}***

***}***

***================================================================***

***Model-2 : Generating implementation object for interface by declaring***

***LambdaExpression***

***Ex:***

***ITest.java***

***package test;***

***public interface ITest {***

***public abstract void m1(int x);***

***}***

***Access.java***

***package test;***

***public class Access {***

***public static ITest getReference()***

***{***

***return (int x)->***

***{***

***System.out.println("===method m1(x)====");***

***System.out.println("The value x:"+x);***

***};***

***}//OuterClass method***

***}//OuterClass***

***DemoModel2.java(MainClass)***

***package maccess;***

***import test.\*;***

***public class DemoModel2 {***

***public static void main(String[] args) {***

***ITest ob = Access.getReference();***

***//This method will create implementation object of ITest***

***ob.m1(12);***

***}***

***}***

***=================================================================***