**Annotations Assignment-7**

1. Create a custom annotation called @Test which can be only applied on a method implying that the following method is a test-case. (Is it possible to restrict the annotation to be applied only on a non-static method?)

Class : Test.java

package org.annotation.app;

import java.lang.annotation.\*;

import java.lang.reflect.\*;

@Retention (RetentionPolicy.*RUNTIME*)

@Target (ElementType.*METHOD*)

@interface Test

{

String str();

}

class First

{

@Test(str="Test Annotation")

public void testCase()

{

}

}

Class : TestMainmethod.java

package org.annotation.app;

import java.lang.reflect.Method;

public class TestMainmethod {

public static void main(String[] args) throws Exception {

First f=new First();

Method m = f.getClass().getMethod("testCase");

Test ts = m.getAnnotation(Test.class);

System.out.println(ts.str());

}

}

1. Build a custom annotation called @Info, which can be used by developers on a class, a property, or a method. The developer can provide the following when using this annotation:
2. AuthorId : <<Developers ID>>-(Mandatory Input)
3. Author : <<Developer name>>-(Optional Input)
4. Supervisor : <<”String Data”>>-(Mandatory Input)
5. Date : <<”String Time”>>-(Mandatory Input)
6. Time : <<Numerical Version>>-(Mandatory Input)
7. Description : <<Description of the class, method, or property>>-(Optional Input)

Class : Author.java

package org.annotation.app;

import java.lang.annotation.Documented;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

@Documented

@Retention (RetentionPolicy.RUNTIME)

@interface info

{

int id();

String name();

String superviser();

String date();

String time();

int version();

}

// Applying annotation.

public class Author {

@info(id = 21, name = "DINESHWARI", supervisor = "Praveen",date="22/12/2021",time="12:15:50 hrs",version=7)

public void display()

{

System.out.println("Hello Barnali");

System.out.println();

}

}

Class : AuthorMainmethod

package org.annotation.app;

import java.lang.reflect.Method;

public class AuthorMainmethod {

public static void main(String[] args) throws NoSuchMethodException, SecurityException {

Author a = new Author();

a.display();

Method m = a.getClass().getMethod("display");

info i = m.getAnnotation(info.class);

System.out.println("Author ID : " +i.id());

System.out.println("Author Name: " +i.name());

System.out.println("Superviser Name: " +i.superviser());

System.***out***.println("Date: " +i.date());

System.***out***.println("Time: " +i.time());

System.***out***.println("Version: " +i.version());

}

}

3.Create a custom annotation called @Execute to be applied on methods. Placing the @Execute method on a method implies that method should be invoked using Reflection API(Invoking the method using Reflection API is out of scope of this assignments). The annotation @Execute should have an optional property “sequence” which can be given values such as 1,2,3…. In the order of priority, In case the sequence property is not used the API may invoke methods in random order.

E.g.

Class MyClass

{

@Execute(Sequence=2)

Public void myMethod1()

{

}

@Execute(Sequence=1)

Public void myMethod2()

{

}

@Execute(Sequence=3)

Public void myMethod3()

{

}

Class : Sequence.java

package org.annotation.app;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target(value = ElementType.*METHOD*)

@Retention(RetentionPolicy.*RUNTIME*)

**@**interface Execute

{

int Sequence();

}

class Sequence

{

@Execute(Sequence=2)

public void method1()

{

System.***out***.println("Method 1");

}

@Execute(Sequence=1)

public void method2()

{

System.***out***.println("Method 2");

}

@Execute(Sequence=3)

public void method3()

{

System.***out***.println("Method 3");

}

}

Class : SequenceMainmethod.java

package org.annotation.app;

import java.lang.reflect.Method;

public class SequenceMainmethod

{

public static void main(String[] args)

{

Sequence s = new Sequence();

Method[] methods = s.getClass().getMethods();

for (Method method : methods)

{

Execute exe = method.getAnnotation(Execute.class);

if (exe != **n**ull)

{

try

{

method.invoke(s);

} catch (Exception e)

{

e.printStackTrace();

}

}

}

}

}

OUTPUT:

Method 2

Method 1

Method 3