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?)

package annotation

import java.lang.annotation.Annotation;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Inherited;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

import java.lang.reflect.Method;

@Inherited

@Documented

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

@interface Test{

String str();

int id();

}

class Anno{

@Test(str="bindu",id=1)

public void Testcase()

{

System.out.println("Annotation first Testcase method");

}

}

public class annotation1 {

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

Anno obj=new Anno();

Method b=obj.getClass().getMethod("Testcase");

Annotation a=b.getAnnotation(Test.class);

System.out.println(((Test) a).str());

System.out.println(((Test) a).id());

}

}

Output:

bindu

1

1. 2. 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)

package annotation;

import java.lang.annotation.Annotation;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Inherited;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Inherited

@Documented

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@interface info{

int id();

String name();

String supervisor();

String date();

String time();

int version();

String description();

}

@info(id=1,name="bindu",supervisor="Bindushree",date="1/1/2022",time="8.45Am",version=9,description="hello everyone")

class annotation{

}

public class annotation2 {

public static void main(String[] args) {

annotation obj=new annotation();

Class b=obj.getClass();

Annotation a=b.getAnnotation(info.class);

info i=(info)a;

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

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

System.out.println("Supervisor:"+i.supervisor());

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

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

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

System.out.println("Description:"+i.description());

}

}

Output:

Author Id:1

Author Name:bindu

Supervisor:Bindushree

Date:1/1/2022

Time:8.45Am

Version:9

Description:hello everyone

3. 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()

{

}

package annotation;

import java.lang.annotation.Annotation;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Inherited;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

@Inherited

@Documented

@Target(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");

}

}

public class annotation3 {

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

sequence obj=new sequence();

Method[] m=obj.getClass().getMethods();

for (Method method : m)

{

Execute a=m.getClass().getAnnotation(Execute.class);

if (a != null)

{

try

{

method.invoke(obj);

} catch (Exception e)

{

e.printStackTrace();

}

}

}

}

}

Output:

Method2

Method1

Method3