**CORE JAVA ASSIGNMENT 7 – ANNOTATIONS**

**Assignment on Annotations**

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?)

**Solution:**

package com.Annotations;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

import java.lang.reflect.Method;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

@interface Test{

}

//Applying annotation

class TestClass{

@Test

public void TestClassMethod(){

System.out.println("This is a TestClassMethod");}

}

public class Main {

public static void main(String[] args) {

TestClass obj = new TestClass();

obj.TestClassMethod();

}

}

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 information when using this annotation:

a) AuthorID: <<Developers ID>> - (Mandatory Input)

b) Author: <<Developers name>> - (Optional Input)

c) Supervisor: <<Developers Supervisor>> - (Optional Input)

d) Date: <<"String Date”>> - (Mandatory Input)

e) Time: <<"String Time”>>- (Mandatory Input)

f) Version: <<Numerical Version >> - (Mandatory Input)

g) Description: <<Description of the class, method, or property >> - (Optional Input)

**Solution:**

package com.annotations2;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target({ElementType.FIELD, ElementType.METHOD, ElementType.TYPE})

@interface info{

String AuthorID() ;

String Author() default "Developer Name";

String Supervisor() ;

String Date() ;

String Time() ;

double Version();

String Description() default "Description of Class,Method,Property";

}

@info(AuthorID = "AS",Supervisor = "HARSH",Date = "8-2-22",Time = "2:30PM" ,Version = 1.0)

class SampleClass{

@info(AuthorID = "AK", Author = "AMAN", Supervisor = "HARSH", Date = "8-2-22", Time = "2:35PM" ,  
Version = 1.0,Description = "This is a property of SampleClass")

String Property;

SampleClass()

{

System.out.println("This Is a SampleClass");

}

@info(AuthorID = "AK", Supervisor = "HARSH", Date = "8-2-22", Time = "2:37PM" ,Version = 1.0,Description = "This is a method of SampleClass")

public void ClassMethod()

{

System.out.println("A method in SampleClass");

}

}

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(){}

Note: The above annotation tells the system that the invocation should be in the order: myMethod2 first, followed by myMethod1 and finally myMethod3

**Solution:**

import javax.sound.midi.Sequence;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

import java.lang.reflect.Method;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

@interface Execute {

int sequence();

}

class MyClass{

@Execute(sequence=2)

public void myMethod1()

{

System.out.println("method1");

}

@Execute(sequence=1)

public void myMethod2()

{

System.out.println("method2");

}

@Execute(sequence=3)

public void myMethod3()

{

System.out.println("method3");

}

}

public class Annotation {

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

MyClass myClass = new MyClass();

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

for (Method method : methods) {

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

if (annos != null) {

try {

method.invoke(myClass);

System.out.println(annos.sequence());

} catch (Exception e) {

e.printStackTrace();

}

}

} } }