**Assignments 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 static method?)

**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() {}

}

**import** java.lang.reflect.Method;

**public** **class** test1 {

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

// **TODO** Auto-generated method stub

first f=**new** first();

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

Test t = m.getAnnotation(Test.**class**);

System.***out***.println(t.str());

}

}

**Output:**  Test Annotation

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 information when using this annotation:
2. AuthorID: <<Developers> - (Mandatory Input)
3. Author: <<Developers name>> - (Optional Input)
4. Supervisor: <<Developers Supervisor>> - (Optional Input)
5. Date: <<”String Date”>> - (Mandatory Input)
6. Time: <<”String Time”>> - (Mandatory Input)
7. Version: <<Numerical Version>> - (Mandatory Input)
8. Description: <<Description of the class, method, or property >> - (Optional Input)

**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 = 123, name = "Jack", superviser = "Sam",date="17/06/2021",time="12:50:50 hrs",version=8)

// importing annotation on a method display()

**public** **void** display()

{

System.***out***.println("Hello Jack");

System.***out***.println();

}

}

**import** java.lang.reflect.Method;

**public** **class** author1 {

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

// **TODO** Auto-generated method stub

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());

}

}

**Output:** Hello Jack

Author ID : 123

Author Name: Jack

Superviser Name: Sam

Date: 17/06/2021

Time: 12:50:50 hrs

Version: 8

1. 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(Sequnce=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

import java.lang.reflect.Method;

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();

}

public 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");

}

}

**import** java.lang.reflect.Method;

**public** **class** sequence1 {

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

// **TODO** Auto-generated method stub

sequence s = **new** sequence();

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

**for** (Method m : methods) {

Execute ex = m.getAnnotation(Execute.**class**);

**if** (ex != **null**) {

**try** {

m.invoke(s);

}

**catch** (Exception e)

{

e.printStackTrace();

}

}

}

}

}

**Output:** Method 2

Method 1

Method 3