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.

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

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:

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](http://i.id/)());

System.out.println("Author Name: " +[i.name](http://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());

}

}

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 = 246, name = "Sam", superviser = "Tom", date="20/05/2021",time="11:30:30 hrs", version=7)

public void display()

{

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

System.out.println();

}

}

Output

Hello Sam

Author ID: 246

Author Name: Sam

Superviser Name: Tom

Date: 20/05/2021

Time: 11:30:30 hrs

Version: 7

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.

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 (value = RetentionPolicy.RUNTIME)

@interface Execute

{

 int Sequence();

}

class Anno

{

@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 AnnotationExecute {

 public static void main(String[] args) {

  // TODO Auto-generated method stub

  Anno ta = new Anno();

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

  for (Method method : methods) {

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

  if (ex != null) {

  try {

  method.invoke(ta);

         }

    catch (Exception e)

    {

    e.printStackTrace();

    }

  }

  }

}

}

Output

Method 2

Method 1

Method 3