Java SE 8 Programming Language

Lab Guides

Document Code	**25e-BM/HR/HDCV/FSOFT**
Version	**1.1**
Effective Date	**20/11/2012**

Hanoi, 06/2019

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1	01/Oct/2018	Add the new labs	Create new	DieuNT1	VinhNV
2	01/Jun/2019	Update template	Fsoft template	DieuNT1	VinhNV

Contents

CODE:	JPL.S.L302
TYPE:	SHORT
LOC:	
DURATION:	30 MINUTES

Unit 6: Advanced OOP – Using Annotation

Knowledge Summary

Annotations are used to provide supplement information about a program.

- Annotations start with '@'.
- Annotations do not change action of a compiled program.
- Annotations help to associate metadata (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.
- Annotations are not pure comments as they can change the way a program is treated by compiler.

There are 3 categories of Annotations:

1. Marker Annotations:

The only purpose is to mark a declaration. These annotations contain no members and do not consist any data.

Example: - @TestAnnotation()

2. Single value Annotations:

These annotations contain only one member and allow a shorthand form of specifying the value of the member.

Example: - @TestAnnotation("testing annotation");

3. Full Annotations:

These annotations consist of multiple data members/ name, value, pairs.

Example:- @TestAnnotation(owner="admin", value="admin")

Built-in Annotations used in custom annotations in java:

Built-in annotation	Usage
@Target	used to specify at which type, the annotation is used (TYPE, FIELD, METHOD, CONSTRUCTOR…).
@Retention	used to specify to what level annotation will be available (Source, Class, Runtime).
@Inherited	By default, annotations are not inherited to subclasses. The @Inherited annotation marks the annotation to be inherited to subclasses.
@Documented	The @Documented Marks the annotation for inclusion in the documentation.

Lab Guide 1: Create custom annotation

Objectives:

- This lab guide helps trainees know how to create a custom annotation in Java.

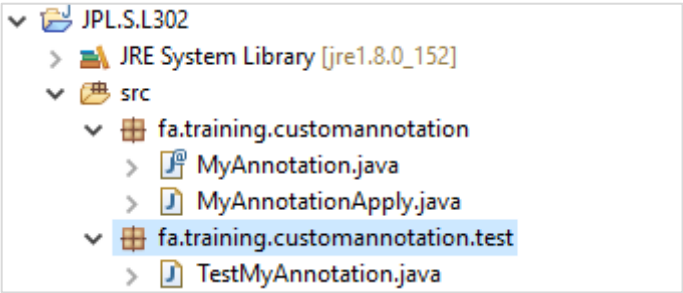
Problem Descriptions:

Create a new package named **fa.training.customannotation** in **JPL.S.L302** project.

- Create a **MyAnnotation** annotation with a method **value()** that returns an integer value.
- Create a Java class named **MyAnnotationApply**.
  - Create a method named **greeting()** that applies the MyAnnotation.
- Create a new package named **fa.training.customannotation.test** in this project.
  - Create a **TestMyAnnotation** class that is used to access the annotation.
  - Run the program to see the output.

Guidelines:

Step 1: Create a project struture like this:



Step 2: Create an annotation named MyAnnotation

```
package fa.training.customannotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Create a custom annotation
 *
 * @author hoabt2
 *
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface MyAnnotation {
        int value();
}

```

Step 3: Create **MyAnnotationApply** class

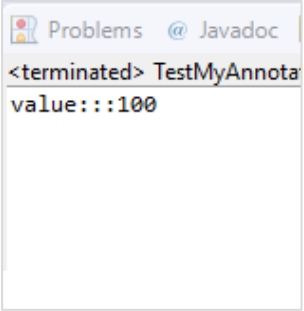
```
20. /**
21.  *
22.  */
23. package fa.training.customannotation;
24.
25. /**
26.  * Show how to apply MyAnnotation
27.  *
28.  * @author hoabt2
29.  *
30.  */
31. public class MyAnnotationApply {
32.
33. @MyAnnotation(value = 100)
34. public void greeting() {
35.         System.out.println("Greeting() from MyAnnotationApply");
36. }
37. }
38.
```

Step 4: Create **TestMyAnnotation** class

```
39. /**
40.  *
41.  */
42. package fa.training.customannotation.test;
43.
44. import java.lang.reflect.Method;
45.
46. import fa.training.customannotation.MyAnnotation;
47. import fa.training.customannotation.MyAnnotationApply;
48.
49. /**
50.  * @author hoabt2
51.  *
52.  */
53. public class TestMyAnnotation {
54.
55. /**
56.  * @param args
57.  * @throws SecurityException
58.  * @throws NoSuchMethodException
59.  */
60. public static void main(String[] args)
61.                     throws NoSuchMethodException, SecurityException {
62.         //Access the annotation
63.         MyAnnotationApply myAnnotationApply = new MyAnnotationApply();
64.         Method method = myAnnotationApply.getClass().getMethod("greeting");
65.         MyAnnotation myAnnotation = method.getAnnotation(MyAnnotation.class);
66.         System.out.println("value:::" + myAnnotation.value());
67. }
68. }
```

Step 5: Run the program to see the output

Run TestMyAnnotation class to see the result:



Lab Guide 2: Access annotation by reflection

Objectives:

- This lab guide helps trainees know how to access annotation using reflection in Java.

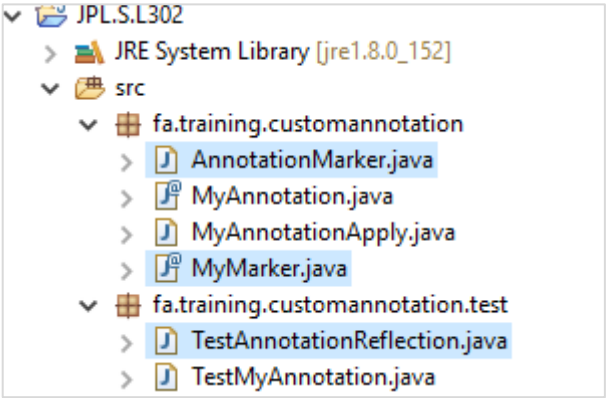
Problem Descriptions:

Add more functions to **JPL.S.L302** project created in Lab Guide 1.

- Create **MyMarker** annotation within **fa.training.customannotation**.
- Create a Java class named **AnnotationMarker** that uses MyMarker to mark certain methods**.**
    - Add some methods to this class and mark them with MyMarker annotation.
- Create a class named **TestAnnotationReflection** within **fa.training.customannotation.test** to calls the corresponding methods using reflection mechanism.
    - Run this class to see the output.

Guidelines:

Step 1: Update project structure like this:



Step 2: Create an annotation named **MyMarker**

```
/**
 *
 */
package fa.training.customannotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Define the annotation
 *
 * @author hoabt2
 *
 */
@Target(value = ElementType.METHOD)
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyMarker {

}

```

Step 3: Create **AnnotationMarker** class that uses MyMarker annotation to mark some certain methods

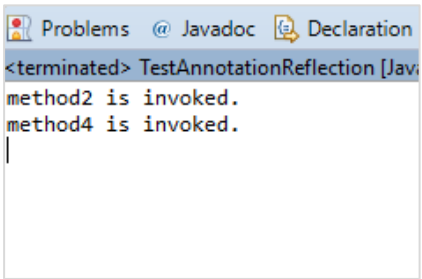
```
package fa.training.customannotation;

/**
 * @author hoabt2
 *
 */
public class AnnotationMarker {

	public void method1() {
		System.out.println("method1 is invoked.");
	}

	@MyMarker
	public void method2() {
		System.out.println("method2 is invoked.");
	}

	public void method3() {
		System.out.println("method3 is invoked.");
	}

	@MyMarker
	public void method4() {
		System.out.println("method4 is invoked.");
	}
	}

```

Step 4: Create **TestAnnotationReflection** class to analyze the annotations and calls the corresponding methods.

```
package fa.training.customannotation.test;

import java.lang.reflect.Method;

import fa.training.customannotation.AnnotationMarker;
import fa.training.customannotation.MyMarker;

/**
 * @author hoabt2
 *
 */
public class TestAnnotationReflection {

      /**
       * @param args
       */
      public static void main(String[] args) {
            AnnotationMarker runner = new AnnotationMarker();
        Method[] methods = runner.getClass().getMethods();

        for (Method method : methods) {
            MyMarker myMarker = method.getAnnotation(MyMarker.class);
            if (myMarker != null) {
                try {
                    method.invoke(runner);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
      }
}
```

Step 5: Run TestAnnotationReflection to see the result

Result:



-- THE END --