Lab 17: Annotations

Objective

In this lab, we will look at some of uses of Annotations in Java.

Overview

In this lab you will:

- Experiment with **@SuppressWarnings** to show the impact of location on the scope of the annotation.
- Create an annotation and see how **@Document** impacts the Javadoc generated for the project.
- Show how **@Target** can constrain where an annotation may be used.

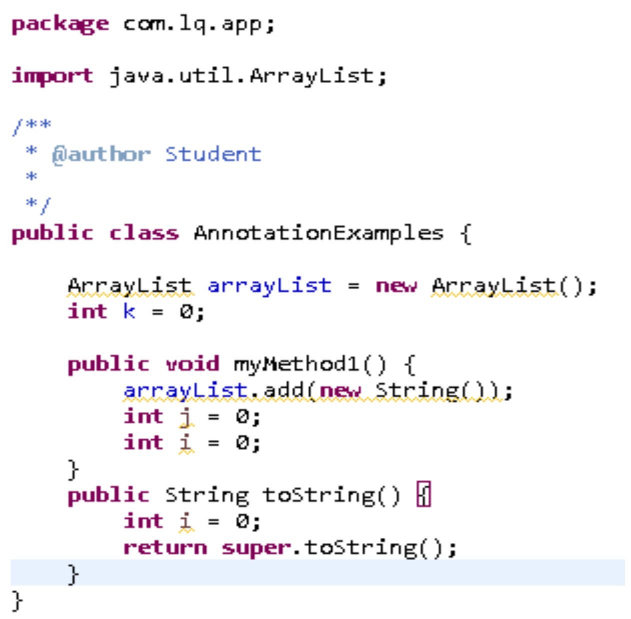
Step by Step Instructions

Exercise 1: @SuppressWarnings

The annotation **@SuppressWarnings** is used to turn off the compiler-generated warnings that crowd the Problems view and clutter the Editor sidebar. The location of the tag determines the scope of the annotation.

1. Create a new Java Project named **Annotations**. In the **src** folder create two packages: **com.lq.annotations** and **com.lq.app**.

2. Create a new class in the **com.lq.app** package named **AnnotationExamples** and implement it as follows:



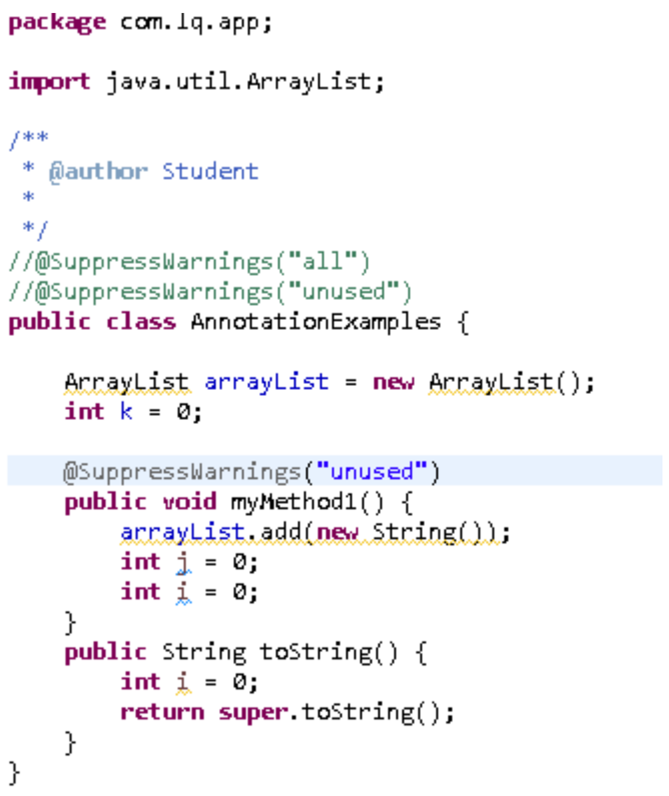
```
package com.lq.app;

import java.util.ArrayList;

/**
 * @author Student
 *
 */
public class AnnotationExamples {

    ArrayList arrayList = new ArrayList();
    int k = 0;

    public void myMethod1() {
        arrayList.add(new String());
        int j = 0;
        int i = 0;
    }
    public String toString() {
        int i = 0;
        return super.toString();
    }
}
```

You will notice several warnings on the sidebar of the Editor view.

3. Add a @SuppressWarnings ("all") annotation after the import statement. Note that all the warnings have been *elided* (hidden from display).

4. Comment out the first annotation (to save it) and add a second immediately following @SuppressWarnings("unused"). Note that class-wide, the warnings for variables declared but not used in code are gone.

5. Comment out the annotation and save your changes. Note that the warnings reappear. Make a copy of the second annotation and paste it immediately before **myMethod1**. Your code should now look similar to the following:

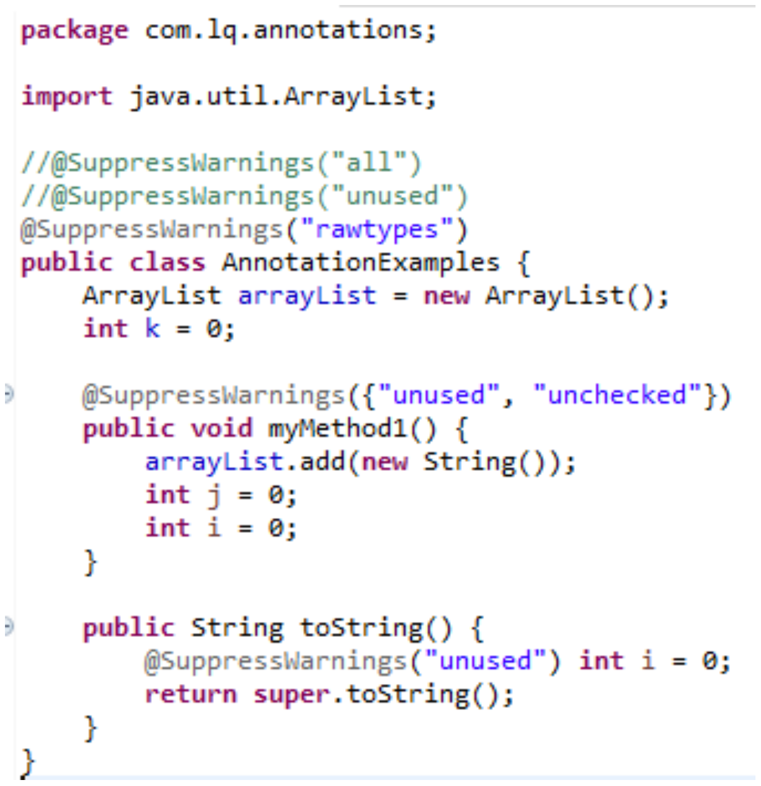


```
package com.lq.app;

import java.util.ArrayList;

/**
 * @author Student
 *
 */
//@SuppressWarnings("all")
//@SuppressWarnings("unused")
public class AnnotationExamples {

    ArrayList arrayList = new ArrayList();
    int k = 0;

    @SuppressWarnings("unused")
    public void myMethod1() {
        arrayList.add(new String());
        int j = 0;
        int i = 0;
    }
    public String toString() {
        int i = 0;
        return super.toString();
    }
}
```

Note that the unused warnings in **myMethod1** are now gone, but the one in the **toString** method is still there. The scope of the annotation is now at the method level.

6. Add a copy of the latest annotation immediately before the **int I = 0**; declaration in the **toString** method. Note that the unused warning for **i** is now gone. This annotation is at the variable level.

7. Add a *type* level (above the class declaration) @SuppressWarnings(“rawtypes”). Add “unchecked” to the existing @SuppressWarnings on the **myMethod** method so that it appears as @SuppressWarnings({“unused”, “unchecked”}). Note that {} are required to pass a list of values to the annotation. You should see that the warnings about generics are now gone. Both warning types needed to be suppressed. Different values for the annotation drive the suppression of warnings of different types.



```
package com.lq.annotations;

import java.util.ArrayList;

//@SuppressWarnings("all")
//@SuppressWarnings("unused")
@SuppressWarnings("rawtypes")
public class AnnotationExamples {
    ArrayList arrayList = new ArrayList();
    int k = 0;

    @SuppressWarnings({"unused", "unchecked"})
    public void myMethod1() {
        arrayList.add(new String());
        int j = 0;
        int i = 0;
    }

    public String toString() {
        @SuppressWarnings("unused") int i = 0;
        return super.toString();
    }
}
```

.

<u>Exercise 2: @Documented and Javadoc</u>

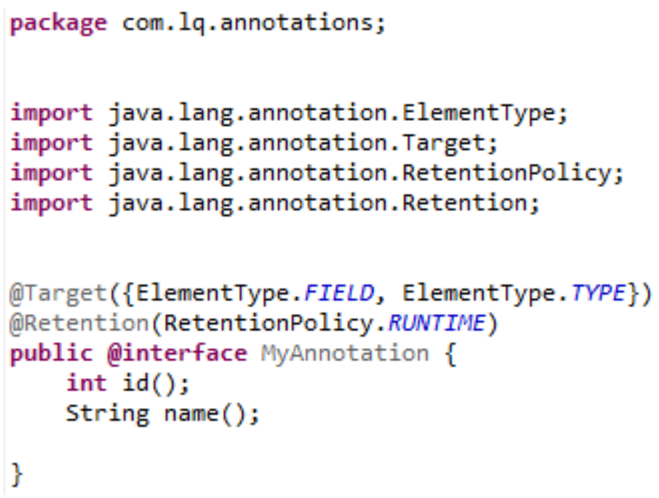
Whether or not an annotation’s values are part of the API Java documentation is determined by the presence of a **@Documented** tag in the declaration.

8. Right-click the **com.lq.annotations** package and select **New** > **Annotation**. The **New Annotation Type** wizard will appear. Name the new annotation **MyAnnotation** and click **Finish**.

9. The new annotation will open in the editor view. Declare two members of the annotation, an **int** method named **id** and a **String** method named **name**. Make sure to declare the RetentionPolicy as RUNTIME, so the Annotation is recorded in the class file by the compiler and retained by the VM at run time by using the @Retention annotation.

When we define an annotation, we sometimes want to specify where that annotation should be used in the code body of our types. The **@Target** meta-annotation provides that mechanism. Use the @Target(ElementType.TYPE) to define that the annotation can applied to another class type.

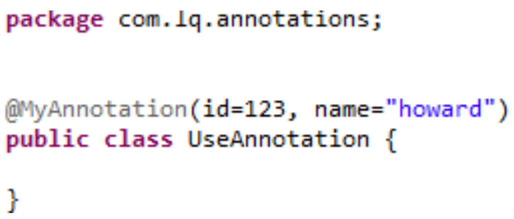
Your annotation should look like the following:



```java
package com.lq.annotations;


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


@Target({ElementType.FIELD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    int id();
    String name();

}
```

10. Create a new class in **com.lq.annotations** package named **UseAnnotation**. Add your new annotation to the class; choose an **id** and **name**.

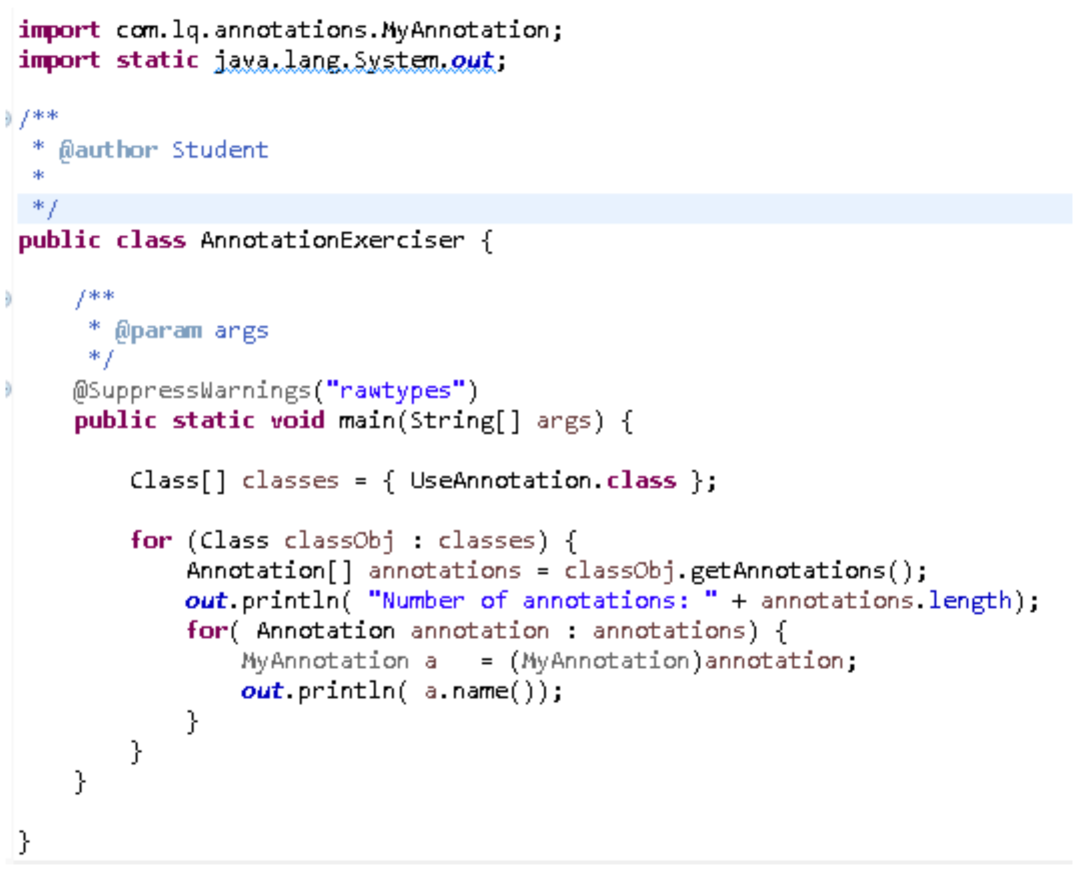
    Your code should look something like the following:



```java
package com.lq.annotations;


@MyAnnotation(id=123, name="howard")
public class UseAnnotation {

}
```

11. Create a new class in the **com.lq.annotations** package (be sure to select the checkbox in the new class wizard that generates a **public static void main(String[] args)** method) named **AnnotationExerciser**. Add the following code within the main method to iterate the annotations defined with the UseAnnotation class. Display the length and the name property values of MyAnnotation type.

    Your annotation code should look like the following:



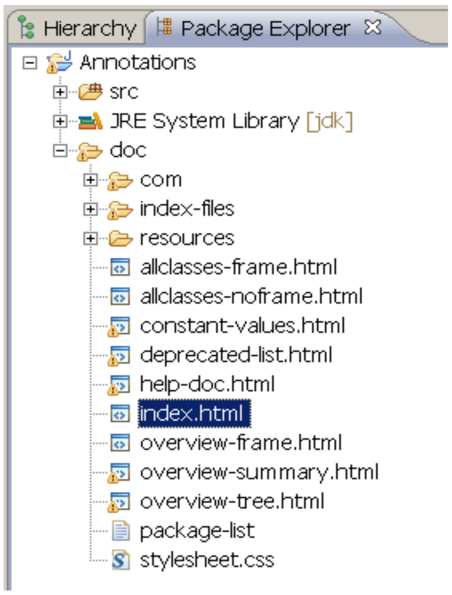
```
import com.lq.annotations.MyAnnotation;
import static java.lang.System.out;

/**
 * @author Student
 *
 */
public class AnnotationExerciser {

    /**
     * @param args
     */
    @SuppressWarnings("rawtypes")
    public static void main(String[] args) {

        Class[] classes = { UseAnnotation.class };

        for (Class classObj : classes) {
            Annotation[] annotations = classObj.getAnnotations();
            out.println( "Number of annotations: " + annotations.length);
            for( Annotation annotation : annotations) {
                MyAnnotation a   = (MyAnnotation)annotation;
                out.println( a.name());
            }
        }
    }

}
```

12. Select **Project** > **Generate Javadoc.** The Generate Javadoc window will appear.

13. Leave all of the settings at their default values and select the Annotations project in the **Select type for which Javadoc will be generated** list. Click **Finish**. In the Console view, you will see a considerable amount of activity as the Javadoc compiler runs.

    The output of the compiler is the generated html that represents the public view of our application's API. Without the **@Documented** annotation in **MyAnnotation**, there is no display of the annotation or its values.

    Note that there is now a new folder in the application. The **doc** folder contains the generated html from the Javadoc compiler.



Right-click **index.html** and select **Open With > Web Browser** to open the Javadoc in the browser. There is no reference to the annotation.

14. Add the **@Documented** meta-annotation to the **MyAnnotation** source and import the required package. **MyAnnotation** should now look like the following:

```
package com.lq.annotations;

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

@Documented
@Target({ElementType.FIELD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    int id();
    String name();

}
```

15. Save your changes and re-generate the Javadoc. You may be asked to confirm the overwrite of the first Javadoc set—be sure to respond **Yes to all** when asked.

16. Re-open index.html to view the Javadoc once again and note the use of **MyAnnotation** is now part of the Javadoc.