Java Annotations

B.Tech. (IT), Sem-6,
Applied Design Patterns and Application Frameworks (ADPAF)

Dharmsinh Desai University
Prof. H B Prajapati

What is annotation?

- Labels applied on source code that can be processed by tools.
- Annotations can be processed at
  - Compile time
  - Run time
  - Deployment time
- Annotations can be applied to
  - Classes
  - Fields
  - Methods
  - Other program elements
- It is like a modifier placed before annotated item.

What is annotation?

- Name of each annotation is preceded by an @ symbol. And it does not end with semicolon.
- By itself the annotation does not do anything. It is processed by tool.
  - Annotations are used by tools to produce derived files such as
    - New java code
    - Deployment descriptor
    - Class files
- Annotations were introduced in J2SE 5.0 platform
  - Annotations like examples in earlier version
    - Transient
    - Serializable interface (it has no method)
    - javadoc comments
    - xdoclet
- Annotation provides a standard, general purpose, more powerful annotation scheme

Why annotation?

- Enables declarative programming style
  - Less coding since tool will generate the boiler plate code from annotations in the source code
  - Easier to change
- No need to maintain "side files"
  - Eliminate the need of deployment descriptor

Java Standard Annotations (part of java.lang package)

Annotations used by compiler

- @Override
  - Override method of base class
- @Deprecated
  - Feature may not be supported in future
- @SuppressWarnings
  - We do not want to see any warning.
  - all: all warnings
  - deprecation: warnings related to deprecation
  - fallthrough: warnings related to missing breaks in switch statements
  - hiding: warnings related to locals that hide variable
  - serial: warnings related to missing serialVersionUID field for a serializable class
  - unchecked: warnings related to unchecked operations
  - unused: warnings related to unused code

Using annotations

- Annotation elements must be compile-time constant values
- May be primitive types: Strings, Classes, enums, annotations, or an array of permitted types
- A list of values is supplied within braces, for example
  - @SuppressWarnings({"unchecked", "deprecation"})
- If there is just element called value, you don't need to specify it (i.e., no need to write value="").

How to define annotations

- Annotations can be defined to have elements. These elements can be processed by the tools that read the annotations.
- Each annotation must be defined by an annotation interface. (@interface)
- Each method declaration in annotation interface defines an element of the annotation.
- Method declarations must not have any parameters or throws clause
- Return types are restricted to primitives, String, Class, enum, annotations, and array of primitive types
- Methods can have default values.

7

Example: annotation

- Example: Test annotation

```
import java.lang.annotation.*;
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Test {
    String id() default "{none}";
    String author() default "{anonymous}";
}
```

8

Example: annotation

- Example: Test annotation
- The @interface declaration creates an actual Java interface.
- Tools that process annotations receive objects that implement the annotation interface.
  - A tool would call the id method to retrieve the id element of a particular Test annotation.
- The Target and Retention annotations are meta-annotations.
  - They annotate the Test annotation,
  - @Target(ElementType.METHOD) indicates that an annotation can be applied to methods only and
  - @Retention(RetentionPolicy.RUNTIME) indicates that the annotation is retained when the class file is loaded into the virtual machine (i.e., runtime).

9

Example: How to use annotation

- Example: Use @Test annotation

```
public class Main{
    @Test(id="1", author="HBP")
    public void myMethod(){
    }
}
```

10

Annotation Processing Tool(apt)

- Apt is a command-line utility for annotation processing.
- It includes a set of reflective APIs and supporting infrastructure to process program annotations.
- First runs annotation processors that can produce new source code and other files.
- Next, apt compiles both original and generated source files.

11

Demo application: AnnotationDemo

- We create a Java application named AnnotationDemo

12

@Override annotation

- Create one class (named as BaseClass), define one method in it.

Start Page | AnnotationDemo.java | BaseClass.java

Source | History

```
1  package annotationdemo;
   public class BaseClass {
       public void testMethod(){
           System.out.println("In BaseClass.testMethod()");
       }
6  }
```

13

@Override annotation

- Create a derived class (named as AnnotationDemo) and redefine the method of base class.

Hovering over this symbol shows information

Start Page | AnnotationDemo.java | BaseClass.java

Source | History

```
1  package annotationdemo;
2  public class AnnotationDemo extends BaseClass{
3
   public void testMethod(){
5  println("AnnotationDemo.testMethod");
6
7  oid main(String[] args) {
8          // TODO code application logic here
9      }
10 }
```

Add @Override Annotation

(Alt-Enter shows hints)

14

@Override annotation

- Now, that warning goes away

Start Page | AnnotationDemo.java | BaseClass.java

Source | History

```
1  package annotationdemo;
2  public class AnnotationDemo extends BaseClass{
3
4      @Override
   public void testMethod(){
6          System.out.println("AnnotationDemo.testMethod");
7      }
8      public static void main(String[] args) {
9          // TODO code application logic here
10     }
11 }
```

15

Using deprecated method

```
1  package annotationdemo;
2
3  import java.util.Date;
4
5  public class AnnotationDemo extends BaseClass{
6      @Override
   public void testMethod(){
8          System.out.println("AnnotationDemo.testMethod");
9      }
10     public static void main(String[] args) {
11         Date d=new Date();
12         d.getDay();
13     }
14 }
```

16

Using deprecated method

```
1  package annotationdemo;
2
3  import java.util.Date;
4
5  public class AnnotationDemo extends BaseClass{
6      @Override
   public void testMethod(){
8          System.out.println("AnnotationDemo.testMethod");
9      }
10     @SuppressWarnings("deprecation")
11     public static void main(String[] args) {
12         Date d=new Date();
13         d.getDay();
14     }
15 }
```

17

Unused variable

```
1  package annotationdemo;
2
3  import java.util.Date;
4
5  public class AnnotationDemo extends BaseClass{
6      @Override
   public void testMethod(){
8          System.out.println("AnnotationDemo.testMethod");
9      }
10     @SuppressWarnings("deprecation")
11     public s(String[] args) {
12         int i;
13         Date d=new Date();
14         d.getDay();
15     }
16 }
```

Variable i is not used

Grey underline symbol for unused variable

18

Class level annotation

- We can use @SuppressWarnings at class level also.

19

Types of annotations

- 3 types of annotations:
  - Marker annotation
    - It has no element
  - Single member annotation
    - It has single element
  - Multi member annotation
    - It has multiple elements

20

Marker annotation

- Annotation without any element
  - Simplest annotation
- Example: @Override
- Define Marker annotation

```
public @interface Initial {}
```

It has no member

- Use Marker annotation (No need to write ())

```
@Initial
public class Main{...}
```

21

Single Member annotation

- It has a single element
  - The element should be named as value.
- How to define

```
public @interface Copyright{
    String value();
}
```

It has a single member

- Usage,
  - If there is only a single element and its name is value, then we do not need to write name of element and = sign while assigning a value

```
@Copyright("2002 ..")
public class SomeClass{}
```

  - If name of a single element is other than value, then we need to write name of element while assigning a value.

22

Multi-member annotation

- This annotation can multiple elements
- Definition

```
public @interface Test {
    String id() default "{none}";
    String author() default "{anonymous}";
}
```

It has a multiple members

- Usage

```
public class Main{
    @Test(id="1", author="HBP")
    public void myMethod(){
    }
}
```

23

Nested annotation

- We define two annotations.
- One annotation is applied to an element of another annotation.
- Definition

```
public @interface Reviewer{
    Name my_name();
    //Name value();
}
public @interface Name{
    String first();
    String last();
}
```

24

Nested annotation

- Usage

```
@Reviewer(my_name=@Name(first="Harshad",
  last="Prajapati"))
public class Main{
  public static void main(String[] args){
  }
}
```

25

Meta Annotation

- Meta annotations are annotations used to annotate other annotations.
  - Examples
  - @Target
  - @Retention

26

@Target meta-annotation

- @Target restricts where we can use the annotation being defined
- Possible locations (defined by Enum ElementType)
  - TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL, VARIABLE, ANNOTATION_TYPE, PACKAGE

27

@Retention meta-annotation

- @Retention meta-annotation indicates how long annotation information is kept.
- Three different possible values (defined by Enum RetentionPolicy)
  - SOURCE: It indicates information will be placed in the source file but will not be available from the class files.
  - CLASS (Default): It indicates that information will be placed in the class file, but will not be available at runtime through reflection.
  - RUNTIME: It indicates that information will be stored in the class file and made available at runtime through reflective APIs.

28

Example using Meta-annotations

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.CLASS)
public @interface Reviewer{
  Name my_name();
}

public @interface Name{
  String first();
  String last();
}
```

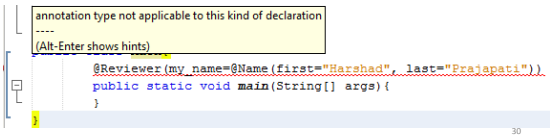
29

Example using Meta-annotations

- We try to use on a method (instead of class)

```
public class Main{
  @Reviewer(my_name=@Name(first="Harshad", last="Prajapati"))
  public static void main(String[] args){
  }
}
```

- We get compile time error



30

Example using Meta-annotations

- Correct way is to use on a class, as Target is TYPE (class, interface, or Enum definition)
- Therefore, the following will not give any error

```
@Reviewer(my_name=@Name(first="Harshad", last="Prajapati"))
public class Main{
    public static void main(String[] args){
    }
}
```

- We do not get compile time error

```
12    @Reviewer(my_name=@Name(first="Harshad", last="Prajapati"))
13    public class Main{
14            public static void main(String[] args){
15            }
16    }
```

31

Example using Meta-annotations

- Suppose, we want to use @Reviewer on both class and method, we need to modify definition of Reviewer annotation as shown in the following:

```
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.CLASS)
public @interface Reviewer{
    Name my_name();
}
```

32

Reflection

- Getting value at runtime
- It is possible only if retention policy is RUNTIME.

33

We define following three annotations

- Marker annotation

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Initial {
}
```

- Single element annotation

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Copyright {
   String value();
}
```

- Multi element annotation

```
@Retention(RetentionPolicy.RUNTIME)
public @interface Author {
   String firstName();
   String lastName();
}
```

34

We apply these three annotations on class and get its information at runtime

```
@Initial
@Copyright("DDU, 2016")
@Author(firstName="Harshad", lastName="Prajapati")
public class Main{
    public static void main(String[] args){
        //Runtime information of marker annotation
        boolean isPresent = Main.class.isAnnotationPresent(Initial.class);
        System.out.println("@Initial Present = "+isPresent);
```

35

We apply these three annotations on class and get its information at runtime

```
        //Runtime information of a single element annotation
        String
    copyrightValue=Main.class.getAnnotation(Copyright.class).value();
        System.out.println("Value of @Copyright = "+copyrightValue);

        //Runtime information of multi-element annotation
        String
    firstNameValue=Main.class.getAnnotation(Author.class).firstName();
        String
    lastNameValue=Main.class.getAnnotation(Author.class).lastName();
        System.out.println("@Author annotation,
    firstName="+firstNameValue+" , lastName="+lastNameValue);
    }
}
```

36

Output

```
run:
@Initial Present = true
Value of @Copyright = DDU, 2016
@Author annotation, firstName=Harshad , lastName=Prajapati
```

37

References

- Video: Java Annotations Tutorial with Programming, VNRgroups.com
- Video: Java Programming – Annotations (from Jpassion.com), Sang Shin

38