Annotations

Annotations

- *Annotations*, a form of metadata, provide data about a program that is not part of the program itself.
- Annotations have no direct effect on the operation of the code they annotate.

Annotations

Annotations have a number of uses, among them:

- **Information for the compiler** — Annotations can be used by the compiler to detect errors or suppress warnings.
- **Compile-time and deployment-time processing** — Software tools can process annotation information to generate code, XML files, and so forth.
- **Runtime processing** — Some annotations are available to be examined at runtime.

The Format of an Annotation

In its simplest form, an annotation looks like the following:

@Entity

Example:

```
@Override
void mySuperMethod() {
    ...
 }
```

The Format of an Annotation

The annotation can include *elements*, which can be named or unnamed, and there are values for those elements:

```
@Author(
    name = "Benjamin Franklin",
    date = "3/27/2003"
)
class MyClass() { ... }
```

or

```
@SuppressWarnings(value = "unchecked")
void myMethod() { ... }
```

The Format of an Annotation

- If there is just one element named value, then the name can be omitted, as in:

```
@SuppressWarnings("unchecked")
 void myMethod() { ... }
```

- If the annotation has no elements, then the parentheses can be omitted, as shown in the previous @Override example.
- It is also possible to use multiple annotations on the same declaration:

```
@Author(name = "Jane Doe")
@EBook class
MyClass { ... }
```

The Format of an Annotation

If the annotations have the same type, then this is called a repeating annotation:

```
@Author(name = "Jane Doe")
@Author(name = "John Smith")
 class MyClass { ... }
```

Repeating annotations are supported as of the Java SE 8 release.

The Format of an Annotation

- The annotation type can be one of the types that are defined in the java.lang or java.lang.annotation packages of the Java SE API.

  In the previous examples, Override and SuppressWarnings are predefined Java annotations.

- It is also possible to define your own annotation type.

  The Author and Ebook annotations in the previous example are custom annotation types.

Where Annotation Can be Used

- Annotations can be applied to declarations: declarations of classes, fields, methods, and other program elements.
- When used on a declaration, each annotation often appears, by convention, on its own line.
- As of the Java SE 8 release, annotations can also be applied to the *use* of types.

```
myString = (@NonNull String) str;
```

Declare an Annotation Type

- Many annotations replace comments in code.
- Suppose that a software group traditionally starts the body of every class with comments providing important information:

```
public class Generation3List extends Generation2List {
    // Author: John Doe
    // Date: 3/17/2002
    // Current revision: 6
     // Last modified: 4/12/2004
     // By: Jane Doe
     // Reviewers: Alice, Bill, Cindy
     // class code goes here
}
```

Declare an Annotation Type

- To add this same metadata with an annotation, you must first define the *annotation type*.
- The syntax for doing this is:

```
@interface ClassPreamble {
      String author();
      String date();
      int currentRevision() default 1;
      String lastModified() default "N/A";
      String lastModifiedBy() default "N/A";
      // Note use of array
      String[] reviewers();
 }
```

Declare an Annotation Type

- The annotation type definition looks similar to an interface definition where the keyword interface is preceded by the at sign (@) (@ = AT, as in annotation type).
- The body of the previous annotation definition contains *annotation type element* declarations, which look a lot like methods.
- Note that they can define optional default values.

Declare an Annotation Type

After the annotation type is defined, you can use annotations of that type, with the values filled in, like this:

```
@ClassPreamble (
      author = "John Doe",
      date = "3/17/2002",
      currentRevision = 6,
      lastModified = "4/12/2004",
      lastModifiedBy = "Jane Doe",
       // Note array notation
      reviewers = {"Alice", "Bob", "Cindy"}
)

public class Generation3List extends Generation2List {
      // class code goes here
}
```

Declare an Annotation Type

To make the information in @ClassPreamble appear in Javadoc-generated documentation, you must annotate the @ClassPreamble definition with the @Documented annotation:

```
// import this to use  @Documented
import java.lang.annotation.*;

@Documented
@interface ClassPreamble {
    // Annotation element definitions
}
```

Predefined Annotation Types-@Deprecated

- **@Deprecated** annotation indicates that the marked element is *deprecated* and should no longer be used.
- The compiler generates a warning whenever a program uses a method, class, or field with the @Deprecated annotation.
- When an element is deprecated, it should also be documented using the Javadoc @deprecated tag.
- The use of the at sign (@) in both Javadoc comments and in annotations is not coincidental: they are related conceptually.
- Note that the Javadoc tag starts with a lowercase *d* and the annotation starts with an uppercase *D*.

Predefined Annotation Types-@Deprecated

```
// Javadoc comment follows
/**
 * @deprecated
 * explanation of why it was deprecated
 */
@Deprecated
static void deprecatedMethod() { }
```

Predefined Annotation Types-@Override

@Override annotation informs the compiler that the element is meant to override an element declared in a superclass.

```
// mark method as a superclass method
// that has been overridden
@Override
int overriddenMethod() { }
```

If a method marked with @Override fails to correctly override a method in one of its superclasses, the compiler generates an error.

Predefined Annotation Types-@SuppressWarnings

@SuppressWarnings annotation tells the compiler to suppress specific warnings that it would otherwise generate.

```
// use a deprecated method and tell
// compiler not to generate a warning
@SuppressWarnings("deprecation")
void useDeprecatedMethod() {
        // deprecation warning
        // - suppressed
        objectOne.deprecatedMethod();
 }
```

Predefined Annotation Types-@SuppressWarnings

Every compiler warning belongs to a category. The Java Language Specification lists two categories: deprecation and unchecked.

- The unchecked warning can occur when interfacing with legacy code written before the advent of generics.
- To suppress multiple categories of warnings, use the following syntax:

```
@SuppressWarnings({"unchecked", "deprecation"})
```

Predefined Annotation Types- @SafeVarargs

- **@SafeVarargs** annotation, when applied to a method or constructor, asserts that the code does not perform potentially unsafe operations on its varargs parameter.
- When this annotation type is used, unchecked warnings relating to varargs usage are suppressed.

Annotations That Apply to Other Annotations -@Retention

@Retention annotation specifies how the marked annotation is stored:

- RetentionPolicy.SOURCE – The marked annotation is retained only in the source level and is ignored by the compiler.
- RetentionPolicy.CLASS – The marked annotation is retained by the compiler at compile time, but is ignored by the Java Virtual Machine (JVM).
- RetentionPolicy.RUNTIME – The marked annotation is retained by the JVM so it can be used by the runtime environment.

Annotations That Apply to Other Annotations -@Documented

@Documented annotation indicates that whenever the specified annotation is used those elements should be documented using the Javadoc tool. (By default, annotations are not included in Javadoc.)

Annotations That Apply to Other Annotations - @Target

@Target annotation marks another annotation to restrict what kind of Java elements the annotation can be applied to. A target annotation specifies one of the following element types as its value:

- ElementType.ANNOTATION_TYPE can be applied to an annotation type.
- ElementType.CONSTRUCTOR can be applied to a constructor.
- ElementType.FIELD can be applied to a field or property.
- ElementType.LOCAL_VARIABLE can be applied to a local variable.
- ElementType.METHOD can be applied to a method-level annotation.
- ElementType.PACKAGE can be applied to a package declaration.
- ElementType.PARAMETER can be applied to the parameters of a method.
- ElementType.TYPE can be applied to any element of a class.

Example

```
package com.test;

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

@Target(ElementType.METHOD)
public @interface TargetTest {
   String hello();
}

public class TargetClass {
   @TargetTest(hello = "abc")
   public void doSomething()
   {
      System.out.println("do something");
   }
}
```

Annotations That Apply to Other Annotations - @Inherited

@Inherited annotation indicates that the annotation type can be inherited from the super class. (This is not true by default.)

When the user queries the annotation type and the class has no annotation for this type, the class' superclass is queried for the annotation type.

This annotation applies only to class declarations.

Repeating Annotations

There are some situations where you want to apply the same annotation to a declaration or type use.
As of the Java SE 8 release, *repeating annotations* enable you to do this.

```
@Schedule(dayOfMonth="last")
@Schedule(dayOfWeek="Fri", hour="23")
public void doPeriodicCleanup() { ... }
```

Step 1: Declare a Repeatable Annotation Type

The annotation type must be marked with the @Repeatable meta-annotation.

Given the custom Schedule example:

```
public @interface Schedule { ... }
```

The code looks like the following:

```
@Repeatable(Schedules.class)
public @interface Schedule { ... }
```

The value of the @Repeatable meta-annotation, in parentheses, is the type of the container annotation that the Java compiler generates to store repeating annotations.

Step 2: Declare the Containing Annotation Type

- The containing annotation type must have a value element with an array type.The component type of the array type must be the repeatable annotation type.
- The declaration for the Schedules containing annotation type is the following:

```
public @interface Schedules {
    Schedule[] value;
}
```