Annotations

Annotations

- Annotations are *metadata*- information about code.
- Prior to annotations (Java 5), we used java comments and Javadoc comments to describe elements of code.
  - We still use these, of course!
- Annotations can often replace these kinds of comments but can do more, too:
  - provide information to the compiler and produce warnings or errors
  - generate code, XML, documentation, or files
    - requires processing by an external tool
  - provide runtime information

Annotation Syntax

- Annotations start with the @ sign
  - Example: @Override
- Annotations can also include *elements* with values
  - Example: @Author (
    name = “Jessica Masters”,
    date = “5/5/2015”)
  - Example: @SuppressWarnings (“unchecked”)
    - only one element, so the name of the element is optional: (value = “unchecked”)
- Multiple annotations can be used in the same declaration
  - By convention, each annotation is placed on their own line.

Annotation Syntax

- Annotations can be applied to **declarations** of:
  - classes
  - fields
  - methods
  - other elements
- As of Java 8, annotations can be applied to the *use* of types

ANNOTATIONS IN THE STANDARD LIBRARY

java.lang Package

- No import needed
- @Deprecated
  - the element should no longer be used
  - generates a compiler warning when that element is used
  - also tag with the Javadoc @deprecated tag, which should explain why the method is deprecated and what to use instead
- @Override
  - method overrides a method from a superclass
  - generates a compiler error if a method header is not correct

java.lang Package

- @SuppressWarnings(“category”)
  - silence compiler warnings from category deprecation or unchecked
    - “unchecked” warnings occur when using pre-generic legacy code
  - @SuppressWarnings(“unchecked”)
  - @SuppressWarnings({“unchecked”, “deprecation”})
- @FunctionalInterface
  - the interface has only one abstract method
  - New to Java 8

Practice

- Use these four annotations on a simple class:
  - Classes: Hirable, Employee, EmployeeTester

WRITING YOUR OWN ANNOTATIONS

Writing Annotations

- Annotations look *kind of like* interfaces.
- They have methods, but:
  - the methods can't have any parameters
  - the method return types can only be: primitive, String, enum, Annotation, or an array of these
  - methods *can* have default values
- Your annotations can have meta-annotations.

Example

```
public @interface MyAnnotation {
    String element1() default “DEFAULT”;
    String element2();
    int element3() default 1;
}
```

Oracle Tutorial Example

```
@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();
}
```

Oracle Tutorial Example

```
@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 {
      …
```

Practice

- Write an annotation to describe information about development.
  - The name of the developer, the version, and the status of the code.
- Use this annotation on a method.
  - Classes: DevelopmentInfo, Employee, EmployeeTester

Meta-Annotations

- Annotations that apply to other annotations
- In the java.lang.annotation package

Meta-Annotations

- @Retention
  - how long the annotation should be stored
  - RetentionPolicy.SOURCE- ignored by compiler and JVM
  - RetentionPolicy.CLASS- retained by compiler, ignored by JVM
  - RetentionPolicy.RUNTIME- retained by the JVM
- @Documented
  - use the annotation in the Javadoc documentation
  - without this tag, annotations are not included in the Javadoc

Meta-Annotations

- @Target
  - restrict the elements to which the annotation can be applied
  - ElementType.ANNOTATION_TYPE
  - ElementType.CONSTRUCTOR
  - ElementType.FIELD
  - ElementType.LOCAL_VARIABLE
  - ElementType.METHOD
  - ElementType.PACKAGE
  - ElementType.PARAMETER
  - ElementType.TYPE

Meta-Annotations

- @Inherited
  - the annotations of this type applied to a parent class will be inherited into the child class
  - by default, they are not inherited
- @Repeatable
  - as of Java 8, you can repeat the same annotation within a single declaration if it is declared as @Repeatable

Example

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD, ElementType.METHOD)
public @interface MyAnnotation {
    String element1() default “DEFAULT”;
    String element2();
    int element3() default 1;
}
```

Practice

- Add meta-annotations to the annotation.

Parsing Annotations

- Use reflection!
- `Method` class methods:
  - isAnnotationPresent(AnnotationType.class)
  - getDeclaredAnnotations()
  - getAnnotation(AnnotationType.class)

Practice

- Write code to print out all annotations of each method in the Employee class.
- Write code to find and invoke all methods that are in testing stage.

A Note about Using Annotations

- These have been small examples about just one way to use annotations **during** development.
- In reality, annotations are used both during **and** after development.
- Annotations can be used any time you need to:
  - describe an element,
  - describe how to process an element, or
  - constrain the use of an element.

Type Annotations

- Pre-Java 8, annotations could only be applied to declarations.
- Now, you can also apply to *type use*.
  - Anywhere you can use a type, you can also use an annotation: the new operator, type casts, implements clauses, and throws clauses
- Type annotations support stronger type checking
- Examples from Oracle:
  - new @Interned MyObject();
  - myString = (@NonNull String) str;
  - class UnmodifiableList<T> implements @Readonly List<@Readonly T> { ... }
  - void monitorTemperature() throws @Critical TemperatureException { ... }

Type Use

- Java 8 didn't include these actual annotations, it just added the ability to use them.
- Other people have to develop the annotations and frameworks to actually use annotations in this way.
- Oracle: "With the judicious use of type annotations and the presence of pluggable type checkers, you can write code that is stronger and less prone to error."
- Example: Checker Framework
  - http://types.cs.washington.edu/checker-framework/