Annotation

Sub-topics of Annotations

- What is and Why annotation?
- How to define and use Annotations?
- 3 different kinds of Annotations
- Meta-Annotations

How Annotation Are Used?

- Annotations are used to affect the way programs are treated by tools and libraries
- Annotations are used by tools to produce derived files
  - Tools: Compiler, IDE, Runtime tools
  - Derived files : New Java code, deployment descriptor, class files

Ad-hoc Annotation-like Examples in pre-J2SE 5.0 Platform

- Ad-hoc Annotation-like examples in pre-J2SE 5.0 platform
  - Transient
  - Serializable interface
  - @deprecated
  - javadoc comments
  - Xdoclet
- J2SE 5.0 Annotation provides a standard, general purpose, more powerful annotation scheme

Why Annotation?

- Enables “declarative programming” style
  - Less coding since tool will generate the boliler plate code from annotations in the source code
  - Easier to change
- Eliminates the need for maintaining "side files" that must be kept up to date with changes in source files
  - Information is kept in the source file
  - example) Eliminate the need of deployment descriptor

Annotation:
How do you define & use annotations?

How to “Define” Annotation Type?

- Annotation type definitions are similar to normal Java interface definitions
  - An at-sign (@) precedes the interface keyword
  - Each method declaration defines an element of the annotation type
  - Method declarations must not have any parameters or a throws clause
  - Return types are restricted to primitives, String, Class, enums, annotations, and arrays of the preceding types
  - Methods can have default values

Example: Annotation Type Definition

```
/**
 * Describes the Request-For-Enhancement(RFE) that led
 * to the presence of the annotated API element.
 */
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date() default "[unimplemented]";
}
```

How To “Use” Annotation

- Once an annotation type is defined, you can use it to annotate declarations
  - class, method, field declarations
- An annotation is a special kind of modifier, and can be used anywhere that other modifiers (such as public, static, or final) can be used
  - By convention, annotations precede other modifiers
  - Annotations consist of an at-sign (@) followed by an annotation type and a parenthesized list of element-value pairs

Example: Usage of Annotation

```
@RequestForEnhancement(
    id       = 2868724,
    synopsis = "Enable time-travel",
    engineer = "Mr. Peabody",
    date     = "4/1/3007"
)
public static void travelThroughTime(Date destination)
   { ... }
```

It is annotating travelThroughTime method

Annotation:
3 Types of Annotations (in terms of Sophistication)

3 Different Kinds of Annotations

- Marker annotation
- Single value annotation
- Normal annotation

Marker Annotation

- An annotation type with no elements
  - Simplest annotation
- Definition

```
/**
 * Indicates that the specification of the annotated API element
 * is preliminary and subject to change.
 */
public @interface Preliminary { }
```

- Usage – No need to have ()

```
@Preliminary
public class TimeTravel { ... }
```

Single Value Annotation

- An annotation type with a single element
  - The element should be named "value"
- Definition

```
/**
 * Associates a copyright notice with the annotated API element.
 */
public @interface Copyright {
    String value();
}
```

- Usage – can omit the element name and equals sign (=)

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

Normal Annotation

- We already have seen an example
- Definition

```
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date();    default "[unimplemented]";
}
```

- Usage

```
@RequestForEnhancement(
    id       = 2868724,
    synopsis = "Enable time-travel",
    engineer = "Mr. Peabody",
    date     = "4/1/3007"
)
public static void travelThroughTime(Date destination) { ... }
```

Annotation:
Meta-Annotations

@Retention Meta-Annotation

- How long annotation information is kept
- Enum RetentionPolicy
  - SOURCE - SOURCE indicates information will be placed i n the source file but will not be available from the class files
  - CLASS (Default)- CLASS indicates that information will be placed in the class file, but will not be available at runtime through reflection
  - RUNTIME - RUNTIME indicates that information will be stored in the class file and made available at runtime through reflective APIs

@Target Meta-Annotation

- Restrictions on use of this annotation
- Enum ElementType
  - TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE, ANNOTATION_TYPE, PACKAGE

Example: Definition and Usage of an Annotation with Meta Annotation

Definition of Accessor annotation

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.CLASS)
public @interface Accessor {
  String variableName();
  String variableType() default "String";
}
```

Usage Example of the Accessor annotation

```
@Accessor(variableName = "name")
public String myVariable;
```

Reflection and Metadata

- Marker annotation

```
boolean isBeta =
  MyClass.class.isAnnotationPresent(BetaVersion.
  class);
```

- Single value annotation

```
String copyright = MyClass.class.getAnnotation
    (Copyright.class).value();
```

- Normal annotation

```
Name author =
  MyClass.class.getAnnotation(Author.class).valu
  e();
String first = author.first();
String last = author.last();
```

Annotation