Java Enterprise Application Development

Lecture 8
Annotations and Reflection

Dr. Fan Hongfei
15 October 2025

Annotations

- A form of metadata, providing data about a program that is not part of the program itself
  - No direct effect on the operation of the code they annotate
- Usage scenarios
  - Information for the compiler
    - Detect errors or suppress warnings
  - Compile-time and deployment-time processing
    - Generate code, XML files, and more
  - Runtime processing

Syntax

- `@Entity`

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

- Can include elements, named or unnamed, with values

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

- Can be applied to declarations of classes, fields, methods, and other program elements
- Each annotation often appears on its own line by convention
- Multiple and repeating annotations on the same declaration are supported

Predefined Annotation Types

- ***@Deprecated:*** the marked element is deprecated and should no longer be used
- ***@Override:*** informing the compiler that the element is meant to override an element declared in a superclass
- ***@SuppressWarnings:*** telling the compiler to suppress specific warnings that it would otherwise generate
- ***@FunctionalInterface***

Declaring an Annotation Type

- 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();
}
```

Introduction to Reflection

- A feature in Java, allowing an executing Java program to **examine or "introspect"** upon itself, and **manipulate** internal properties of the program
  - Examining properties of a class
  - Setting and getting field values
  - Invoking methods
  - ...
- Powerful, and has no equivalent in many other languages

Introduction to Reflection (cont.)

- Example

```java
public static void main(String[] args) throws
ClassNotFoundException {
    Class<?> c = Class.forName(args[0]);
    Method[] m = c.getDeclaredMethods();
    for (Method method : m) {
        System.out.println(method.toString());
    }
}
```

- Loading the specified class, and retrieving the list of methods defined in the class
- *`java.lang.reflect.Method`* is a class representing a method

Entry Point for All Reflection APIs

- JVM instantiates an immutable instance of *`java.lang.Class`* for every type of object
  - Providing methods to examine the runtime properties of the object including its members and type information
  - Providing the ability to create new classes and objects
- Retrieving class objects
  - *`Object.getClass()`*
  - *`.class`*
  - *`Class.forName()`*
  - *`Class.getSuperclass()`*
  - *`Class.getClasses()`*
  - …

Reflection APIs

- Examine class modifiers and types
  - *getModifiers()*
  - *getTypeParameters()*
  - *getGenericInterfaces()*
  - *…*
- Fetch annotation information
  - *getAnnotations()*
- Discover class members
  - *getDeclaredFields() / getDeclaredMethods():* including private members, but no inherited members
  - *getFields() / getMethods():* including inherited members, but no private members
  - …

Reflection APIs (cont.)

- Field
  - Providing methods for accessing type information and setting and getting values of a field on a given object
  - *`getModifiers()`*
  - *`getType()`*
  - *`get(Object obj), getByte(Object obj), getInt(Object obj), ...`*
  - *`set(Object obj), setByte(Object obj), setInt(Object obj), ...`*

Reflection APIs (cont.)

- Method
  - Providing APIs to access information about a method's modifiers, return type and parameters, and to invoke methods
  - *getReturnType()*
  - *getGenericReturnType()*
  - *getParameterTypes()*
  - *invoke(Object obj, Object... args)*

Drawbacks

- Performance suffers
  - Reflective method invocation is much slower than normal method invocation
- Security restrictions
  - Reflection requires a runtime permission which may not be present in a restricted security context
- Readability sacrificed
  - The code required to perform reflective access is clumsy and verbose
- Exposure of internals
  - May result in unexpected side-effects
- Lose benefits of compile-time type checking
- **Therefore, use reflection only when necessary**

Further Reading

- https://docs.oracle.com/javase/tutorial/reflect/TOC.html
- https://www.oracle.com/technical-resources/articles/java/javareflection.html
- Items 39-41 in *Effective Java (3rd Edition)*