Programming in Java

7. Java Annotations

Java Annotations

- An annotation associates, or annotates, metadata with a program elements
  - May be a module, a package, a class, an interface, a field of a class, a local variable, a method, a parameter of a method, an enum, an annotation, a type parameter
- An annotation is used as a “modifier” in a declaration of a program element like any other modifiers like public or static
- Unlike a modifier, an annotation does not modify the meaning of the program elements.
- It provides metadata the program element that it annotates.
- Annotations provide information to the Java compiler as to the role an element plays
  - Like the @Entity annotation or the @Override annotations
  - Allows for more robust error checking
  - Offloads a lot of boilerplate code to the compiler

Built in Annotations

- There are number of built-in annotations
  - Many can take arguments

Annotation	Purpose	Example
`@Override`	Ensures method overrides a superclass	`@Override public void run() {}`
`@Deprecated`	Marks method as outdated	`@Deprecated public void oldMethod()`
`@SuppressWarnings`	Suppresses compiler warnings	`@SuppressWarnings("unchecked")`
`@FunctionalInterface`	Marks interface with single method	`@FunctionalInterface interface MyFunc`
`@SafeVarargs`	Suppresses warnings for varargs generics	`@SafeVarargs static void demo(T... t)`

Annotations Functions

- Annotations serve a number of functions

Purpose	Example
Compiler instructions	`@Override`, `@Deprecated`
Runtime processing	`@Retention(RUNTIME)`
Configuration	JPA, Spring, JUnit annotations
Code analysis and tooling	Used by IDEs, Checkers

Compiler Processing

- Provides additional information to the compiler
- For example, the @Override annotations
  - Tell the compiler a method definition overrides another
  - Allows the compiler to confirm the override is done correctly
  - Otherwise a error in the method signature would not be an override
  - But it would still be legal but create odd runtime behavior

Configuration

- We have already seen examples of this
  - In the JPA section, we saw the use of @Entity and @ID for example
  - This allow the framework to map Java elements to structural roles
  - The example shows the Java class and method mapped to specific structural roles in a REST controller used in a web service

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController  // Marks this class as a REST controller
public class HelloController {

    @GetMapping("/hello")  // Maps HTTP GET requests to /hello
    public String sayHello() {
        return "Hello, world!";
    }
}
```

Custom Annotations

- Declaring an annotation type is similar to declaring an interface type
  - An annotation type declaration is a special kind of interface type declaration.
  - The interface keyword, which is preceded by the @ sign (at sign) declares an annotation type.

```
[modifiers] @ interface <annotation-type-name> {
    // Annotation type body goes here
}
```

  - The modifiers are public, etc
  - Static and default methods in interface types, not allowed in annotation types.
  - Static and default methods are meant to contain some logic which is not metadata

Custom Annotations

- Annotations need to be compiled
  - It will produce a corresponding class file
  - Annotations have to be imported like any other types
- Annotations can have data
  - Defined like a an abstract method in an interface
  - Values are provided with keyword value pairs when the annotation is used

```java
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;

// This annotation can be applied to methods and is retained at runtime
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Info {
    String author();
    String date();
    int revision() default 1; // Optional element with default value
}
```

```java
public class Demo {

    @Info(author = "Rod Davison", date = "2025-06-21", revision = 2)
    public void sampleMethod() {
        System.out.println("This is an annotated method.");
    }
}
```


Lab 7-1
Custom Annotation


Java™