Java Code Annotations

Week 4 Presentation 2

Annotations

- You may have noticed some annotations already; it's common when you use inheritance and extend an abstract class. Here you redefine in the child class a method defined in the parent class to precede the child class definition with @Override (which means *replace* the existing method).
- This is an annotation, which is completely optional but warns javac of your intent. If you mistype the name, it will enable javac to detect an error if there is no such method in the parent class.

Annotations (TAGS)

- Completely optional
  Change nothing to what the program does
- Help Javac – or the program
- Much used by code-generating tools

As annotations can be accessed by programs, many tools that generate code – for tests, for instance – use annotations to collect information they cannot get otherwise.

Annotations

Annotations can take different forms, from the simple marker to some kinds of function calls that are outside the program itself.

- Marker e.g. @Override
- Single parameter

@Select (sql ="SELECT userId, firstName,…

- Multiple parameters

Metadata

= DATA about the CODE

Metadata is a big concern in real life. Companies consider programs as assets, on which several generations of developers can work, which must be written in an easy-to-comprehend, standard way, and well documented. Metadata allows, among many other things, to industrialize code production and to standardize everything.

Standard Annotations

Java provides three standard annotations, which are all a way to give hints to javac.

- @Override
- @Deprecated = Old / Obsolete
- @SuppressWarnings

e.g.

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

2 Annotations were added in Java 7

- @SafeVarargs
- @FunctionalInterface

"Varargs" stands for "Variable [number of] Arguments"

FunctionalInterface marks a functional interface as we saw can be assigned to anonymous functions

Making Annotations

- You can create your own annotations! Declared as interfaces

```
import java.lang.annotation.*;

public @interface MyAnnotation {
 }
```

Annotation-based tools use their own set of annotations, which you just need to import before using.

Making Annotations

- As annotations are a bit special (it's a kind of program in the program) they are constrained by a number of rules.

- Methods should not have any parameters.
- Methods declarations should not have any throws clauses.
- *Return type must be one of:*
  - primitive type, String, enum, Class or an array of one of these

https://docs.oracle.com/javase/tutorial/java/annotations/

Custom Annotations Example: Structured Documentation

```
class SomeClass {
// Created by S Faroult // Creation date: 21/03/2017
// Revision history:
//
// 24/05/2017–Constructor with String parameter
// 26/02/2018 –   toString() rewritten
```

What can you use annotations for in practice? Any Software Development Manager dreams of seeing comments like this. But every developer will not write them, and those who do may use a different format.

Custom Annotations Example: Structured Documentation

```
import java.lang.annotation.*;

public @interface ClassDoc {
        String author();
        String created();
        String[] revisions();
}
```

ClassDoc.java

Annotations could help turning readable but unparsable comments into usable data…

Custom Annotations Example: Structured Documentation

```
@ClassDoc (     author="S Faroult",
                created="21/03/2017",
                revisions={"24/05/2017 – Constructor
                with String parameter",
                "26/02/2018 – toString() rewritten"})
class SomeClass {
```

As the annotation is defined and checked by javac it becomes possible to ensure a standard way for documenting the code. This information can then be retrieved to document programs (such as with Javadoc – will cover later).

Meta Annotations (Annotations about Annotations)

- **@Retentions:** says whether the annotation is available to javac or available at runtime.
- **@Documented**: Make it appear in the docs generated by Javadoc tool.
- **@Target**: What it applies to: Constructor, Method, Parameter…
- **@Inherited**: Passed to child classes (false by default).
- **@Repeatable**: Can be applied multiple times.

Testing JUnit

- JUNIT generates tests for checking your programs. Frameworks are software tools that try to generate automatically the boring bits of a program (which are often a lot of copy-and-paste).

https://junit.org/junit5/

@Test

@BeforeClass – Run once before any of the test methods in the class, public static void
@AfterClass – Run once after all the tests in the class have been run, public static void
@Before – Run before @Test, public void
@After – Run after @Test, public void
@Test – This is the test method to run, public void