Technische Hochschule Rosenheim
Technical University of Applied Sciences

Object-oriented programming
Chapter 7b – Annotations

Prof. Dr Kai Höfig

JUnit and annotations

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- We already learned about some annotations in our test classes, e.g. `@Test`
- In this chapter, we will begin by focusing on the topic of annotations in general

  - Use of annotations as information for the compiler
  - Detection of errors at compile time
  - Predefined annotations of the Java API
  - Definition (declaration) of annotations
  - Meta annotations

- Then we will take a look at the annotations of the JUnit5 test framework

  - `@Test`
  - `@BeforeEach`
  - `@BeforeAll`
  - `@AfterEach`
  - `@AfterAll`

Annotations

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Annotations are meta information, i.e. information about information**
  - Special syntax element: @, e.g. `@Override`, `@Test`
  - Can be added to classes, attributes, methods and arguments.
  - Provide additional semantic information about the programme.
  - Semantics are not processed directly by the compiler, i.e. they have no direct influence on the programme execution
  - Annotations affect the semantics of "things" that use these elements

- **Information for the compiler**
  - Detect programming errors, generate warnings

- **Information for tools (toolkits)**
  - For toolkits that work on the source code
  - At compile time, e.g. for automatic code generation
  - At deployment time, e.g. for verification of requirements
  - At runtime, e.g. for analysing objects.

- **Examples of annotations**
  - Predefined annotations, for language features and frequent use; object-relational mapping of storing objects with the Jakarta Persistence API (JPA); test frameworks testing with JUnit 5

Predefined annotations: `@Override`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Typical error: **overloading** instead of **overwriting**

```
class MyClass {
  // false, but not detected
  public boolean equals(MyClass m) { /* ... */ }

  // this way would have been correct!
  public boolean equals(Object o) { /* ... */ }
}
```

- Through annotation with `@Override`, the compiler can detect errors:

```
class MyClass {
  @Override
  public boolean equals(MyClass m) { /* ... */ }  // compiler error!

  @Override
  public boolean equals(Object o) { /* ... */ } // ok
}
```

Predefined annotations: `@SuppressWarnings`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Current Java compilers generate a lot of warnings
- Many are justified and point out programming errors
- However, some are unavoidable sometimes
- Turn off warnings & document at the same time
- Type of warning to be suppressed is parameter (and compiler-specific)

```
@SuppressWarnings("unchecked")
public void methodWithScaryWarning() {
    List rawList = new ArrayList();

  // this would normally give a warning:
  List<String> stringList = (List<String>)rawList;
}
```

Predefined annotations: `@Deprecated`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Note to the programmer not to use a method/class/etc. (anymore)
- Often when updating toolkits or libraries
- Warning when using elements that are annotated with @Deprecated

```
class OldClass {
  @Deprecated
  public static void stupidOldMethod() { /* ... */ }
}
class NewClass {
  public boolean myNewMethod() {
        stupidOldMethod();  // compiler warning
  }
}
```

Definition (declaration) of annotations

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Annotations are defined (declared) in a similar way to interfaces, but with the @ sign in front of the interface keyword:
  - In the declaration of an annotation, methods can be declared that describe elements of the annotation.
  - Methods in annotations have no parameters.
  - Allowed return types: byte, short, int, long, float, double, String, Class, Annotation and Enumeration, as well as fields about these types
  - Definition of default values is possible.

- **Example:** Annotation @BugFix to show who tried to fix which bug and when.
  - Here as method annotation
  - Third argument (bugsFixed) is empty → default value ("")!

```
public @interface BugFix {
    String author();
    String date();
    String bugsFixed() default "" ;
}
```

```
@BugFix(author="max", date="04.05.2018")
public void greatFunction() {
  // ...
}
```

Writing simplifications

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Annotations without methods: marker annotations
  - Round brackets can be omitted
  - Examples: `@Override`, `@Deprecated`
- Annotations with exactly one method: value annotations
  - Only a single method called `value`
  - Identifier `value` can be omitted when used

```
public @interface ReleaseVersion {
    String value() ;
}
```

```
@ReleaseVersion("1.2.5")
public class UseAnnotations {
}
```

Annotations for JUnit5

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **`@Test`**
  Marks a method as a test method → execution as a test case in IntelliJ or Gradle test.

- **`@BeforeEach`, `@BeforeAll`, `@AfterEach`, `@AfterAll`**
  Marks methods that are to be executed before or after test cases, e.g. in order to initialise data structures.

- **`@Disabled`**
  Marks a test case as to be ignored.

```
class MyTestCollection {
  @BeforeAll
  static void initAll() {
    /* is executed once before all */ }

  @BeforeEach
  void init() {
    /* is executed again before each test */ }

  @Test
  void aTestCase() {  }

  @AfterEach
  void tearDown() {
    /* is executed again after each test */ }

  @AfterAll
  static void tearDownAll() {
    /* is executed once at the end */ }
}
```

Testing with JUnit5

Technische Hochschule Rosenheim
Technical University of Applied Sciences

1. Create a new test class
2. Use @Test to annotate test case methods
3. Helper methods for testing
   https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions

- `assertEquals(expected, actual)`
- `assertTrue(actual)`
- `assertFalse(actual)`
- `assertThrows(exception, lambda)`
- `assertArrayEquals(expected, actual)`

```
@Test
void exceptionTesting() {
    Throwable exception = assertThrows(IllegalArgumentException.class,
            () -> {
        throw new IllegalArgumentException("a message");
            });
  assertEquals("a message", exception.getMessage());
}
```

Examples of toolkits that use annotations

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- JUnit 5: Automated testing
- Jakarta Persistence API: De/Serialisation of objects
- Google Gson: De/Serialisation of objects to/from JSON
- https://spring.io/: Web applications
- http://square.github.io/retrofit/: REST API Adapter
- http://jakewharton.github.io/butterknife/: Android GUI
- https://github.com/chalup/microorm/: ORM