Technische Hochschule Rosenheim
Technical University of Applied Sciences

Objektorientierte Programmierung
Kapitel 7b – Annotationen

Prof. Dr. Kai Höfig

JUnit und Annotationen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Einige Annotationen haben wir in unseren Testklassen bereits kennengelernt, z.B. `@Test`
- In diesem Kapitel widmen wir uns zunächst ganz allgemein dem Thema Annotationen
  - Verwendung von Annotationen als Hinweise für den Compiler
  - Entdecken von Fehlern zur Compilezeit
  - Vordefinierte Annotationen der Java API
  - Definition von Annotationen
  - Meta Annotationen
- Dann werfen wir einen Blick auf die Annotationen des JUnit5 Testframeworks
  - `@Test`
  - `@BeforeEach`
  - `@BeforeAll`
  - `@AfterEach`
  - `@AfterAll`

Annotationen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Annotationen sind Metainformationen, d.h. Informationen über Informationen**
  - Spezielles syntaktisches Element: @, z.B. `@Override`, `@Test`
  - Können an Klassen, Attribute, Methoden und Argumente geheftet werden.
  - Stellen zusätzliche semantische Informationen zum Programm bereit.
  - Semantik wird vom Compiler nicht direkt bearbeitet, d.h. sie haben keinen direkten Einfluss auf den Programmablauf
  - Annotationen beeinflussen die Semantik von "Dingen," die diese Elemente verwenden

- **Informationen für den Compiler**
  - Programmierfehler erkennen, Warnungen erzeugen

- **Informationen für Werkzeuge (Toolkits)**
  - Für Toolkits, welche auf dem Quelltext arbeiten
  - Zum Übersetzungszeitpunkt (engl. compile time), z.B. zur automatischen Codegenerierung
  - Zum Deploymentzeitpunkt (engl. deployment time), z.B. zur Verifizierung von Anforderungen
  - Zur Laufzeit (engl. run time), z.B. zum analysieren von Objekten.

- **Beispiele für Annotationen**
  - Vordefinierte Annotationen, für Sprachfeatures und häufige Verwendung; Objekt-relationale Abbildung Speicherung von Objekten mit der Java Persistence API (JPA); Testframeworks Testen mit JUnit 5

Vordefinierte Annotationen: `@Override`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Typischer Fehler: **überladen** statt **überschreiben**

```
class MeineKlasse {
    // falsch, aber unentdeckt
    public boolean equals(MeineKlasse m) { /* ... */ }

    // so wäre es richtig gewesen!
    public boolean equals(Object o) { /* ... */ }
}
```

- Durch die Annotation mit `@Override` kann der Compiler Fehler erkennen:

```
class MeineKlasse {
    @Override
    public boolean equals(MeineKlasse m) { /* ... */ }  // Compilerfehler!

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

Vordefinierte Annotationen: `@SuppressWarnings`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Aktuelle Java Compiler erzeugen sehr viele Warnungen
- Viele sind berechtigt und weisen auf Programmierfehler hin
- Einige sind jedoch manchmal unvermeidbar
- Warnungen abschalten & gleichzeitig dokumentieren
- Art der zu unterdrückenden Warnung ist Parameter (und compilerspezifisch)

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

    // das gäbe normalerweise eine Warnung:
    List<String> stringList = (List<String>)rawList;
}
```

Vordefinierte Annotationen: `@Deprecated`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Hinweis an den Programmierer, eine Methode/Klasse/etc. nicht (mehr) zu verwenden
- Oft bei Update von Toolkits bzw. Bibliotheken
- Warnung bei Verwendung von Elementen, die mit @Deprecated annotiert sind

```
class AlteKlasse {
    @Deprecated
    public static void doofeAlteMethode() { /* ... */ }
}
class NeueKlasse {
    public boolean meineNeueMethode() {
        doofeAlteMethode();  // Compiler Warning
    }
}
```

Definition von Annotationen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Annotationen werden ähnlich wie Interfaces definiert, jedoch mit dem @-Zeichen vor dem Schlüsselwort Interface:
  - In der Definition einer Annotation können Methoden deklariert werden, die Elemente der Annotation beschreiben.
  - Methoden in Annotationen besitzen keine Parameter.
  - Erlaubte Rückgabetypen: byte, short, int, long, float, double, String, Class, Annotation und Enumeration sowie Felder über diese Typen
  - Definition von Defaultwerten möglich.

- **Beispiel:** Annotation @BugFix um anzuzeigen, wer wann welchen Fehler zu beheben versucht hat.
  - Hier als Methodenannotation
  - Drittes Argument (bugsFixed) ist leer → Defaultwert (""")!

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

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

Schreibvereinfachungen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Annotationen ohne Methoden: Markerannotationen
  - Runde Klammern können entfallen
  - Beispiele: `@Override`, `@Deprecated`

- Annotationen mit genau einer Methode: Value-Annotationen
  - Nur eine einzige Methode mit Namen `value`
  - Bezeichner `value` kann bei Verwendung weggelassen werden

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

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

Annotationen für JUnit5

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **`@Test`**
  Markiert eine Methode als Testmethode → Ausführung als Testcase in IntelliJ oder gradle test.

- **`@BeforeEach, @BeforeAll, @AfterEach, @AfterAll`**
  Markiert Methoden, welche vor oder nach Testcases ausgeführt werden soll, um z.B. Datenstrukturen zu initialisieren.

- **`@Disabled`**
  Markiert einen Testcase als zu ignorieren.

```
class MeineTestSammlung {
    @BeforeAll
    static void initAll() {
        /* wird einmal vor allen ausgeführt */ }

    @BeforeEach
    void init() {
        /* wird vor jedem Test erneut ausgeführt */ }

    @Test
    void einTestFall() {  }

    @AfterEach
    void tearDown() {
        /* nach jedem Test erneut ausgeführt */ }

    @AfterAll
    static void tearDownAll() {
        /* wird einmal am Ende ausgeführt */ }
}
```

Testen mit JUnit5

Technische Hochschule Rosenheim
Technical University of Applied Sciences

1. Neue Testklasse erstellen
2. Testcasemethoden mit @Test annotieren
3. Hilfsmethoden zum Testen
   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());
}
```

Beispiele für Toolkits, welche Annotationen verwenden

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- JUnit5: Automatisiertes Testen
- Java Persistence API: De/Serialisierung von Objekten
- Google Gson: De/Serialisierung von Objekten nach/von JSON
- https://spring.io/: Webapplikationen
- http://square.github.io/retrofit/: REST API Adapter
- http://jakewharton.github.io/butterknife/: Android GUI
- https://github.com/chalup/microorm/: ORM