TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Fakultät Informatik, Lehrstuhl für Softwaretechnologie

Übung Softwaretechnologie 2
Lösungshinweise zum Komplex 3

WS 2023/24, Dr. Dmytro Pukhkaiev

Schritt 3: Maven-Projekt initialisieren (1/2)

- Viele Möglichkeiten der Erstellung
  - IDE-Wizards
  - Maven Archetypes (**mvn archetype:generate …**)
  - Template nutzen (z.B. aus JUnit-5-Dokumentation)
- bei einem Git-Repository: Eintrag von **target/** in **.gitignore** nicht vergessen
- Ausführung auf Kommandozeile: **mvn compile** und **mvn test**
- Maven-Goals sind nach Import in IDE auch dort verfügbar
- Achtung: Änderungen an pom.xml müssen mit IDE meist explizit synchronisiert werden!

Schritt 3: Maven-Projekt initialisieren (2/2)

- Unterschied Dependencies und Plugins
  - Plugins – Erweiterungen für das Build-Tool Maven
  - Dependencies – Bibliotheken, die der Code nutzt (**CLASSPATH**)
- **<dependencyManagement/>** und **<pluginManagement/>** können zum Festlegen von Versionen genutzt werden, ohne die eigentliche Abhängigkeit bzw. das Plugin einzubinden
- **<dependencies/>** und **<plugins/>** hingegen binden tatsächlich die Abhängigkeit bzw. das Plugin ein und konfigurieren letzteres, wenn nötig (Element **<configuration/>**)
- Häufiger Fehler: Jacoco wurde nur in **<pluginManagement/>** eingetragen

Schritt 4: Tests mit JUnit 5 und AssertJ

- Immer die offiziellen Anleitungen beachten – dort gibt es wichtige Hinweise!
- **maven-surefire-plugin** muss mindestens Version 2.22.2 haben und sollte daher explizit angegeben werden – Standardwert ist systemabhängig, z.B. von lokaler Maven-Version
- **<scope>test</scope>** sollte für Abhängigkeiten angegeben werden
- Warum AssertJ oder Google Truth?
  - nur ein statischer Import notwendig: **assertThat**
  - sehr zugänglich über Code-Completion: **assertThat(sth).<TAB>**
  - nützlichere Fehlermeldungen im Falle von Assertion Failures
  - natürliche Leserichtung
- Beispiele:
  - **assertEquals(list.size(), 0) –> assertThat(list).isEmpty()**
  - **assertTrue(list.contains(elem)) –> assertThat(list).contains(elem)**

Schritt 5: Jacoco und Eclemma

- Jacoco: Coverage-Engine mit Java-Agent
- Eclemma: Eclipse-Plugin (z.B. aus Marketplace), das Jacoco nutzt
- Arten der Abdeckung
  - *instruction coverage* / Anweisungsabdeckung (auf Bytecode-Ebene)
  - *line coverage* / Zeilenabdeckung (Debug-Symbole müssen im Bytecode sein)
  - *branch coverage* / Zweigabdeckung
- Konfiguration als **<plugin>** (**org.jacoco** / **jacoco-maven-plugin**)
  - Bindung des **jacoco:{prepare-agent,report}** Goals an die **test**-Phase mit **<executions/>**
  - Äquivalent zum Aufruf **mvn jacoco:prepare-agent test jacoco:report**

Schritt 6: Fehler im Code

In Methode **add(E o)** wird immer auf **end.next** zugegriffen, auch wenn **end == null**.
Der if-Test **end != null** hat keine Auswirkungen, da danach ein leeres Statement kommt.

```
if (end != null); {
    end.next = e;
}
```

ist formatiert:

```
if (end != null)
    ;
{
    end.next = e;
}
```

In der Iteratormethode **next()** wird das nächste statt des aktuellen Elements zurückgegeben und am Ende der Iteration fehlt die **NoSuchElementException** (siehe Javadoc von **Iterator<E>**).
Letzterer Fehler fällt spätestens mit SpotBugs auf.

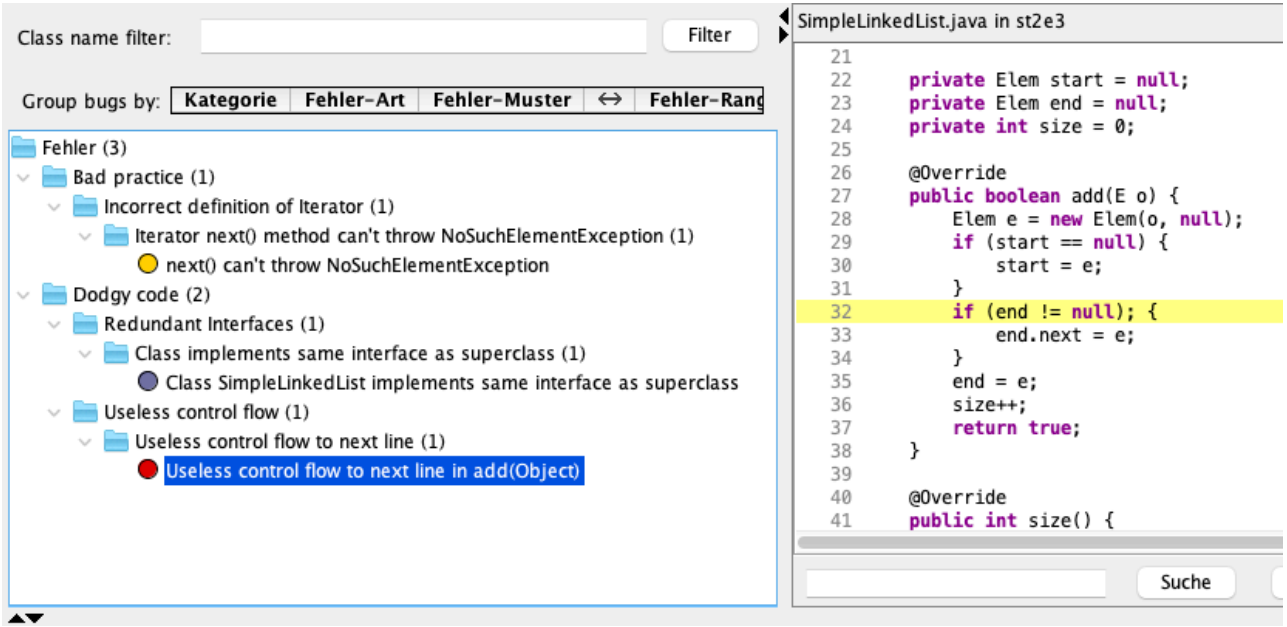
Schritt 7: Statische Codeanalysen (1/2)

Compiler-Flag **-Xlint:all** muss im **maven-compiler-plugin** konfiguriert werden:

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <showWarnings>true</showWarnings>
    <compilerArgs>
      <arg>-Xlint:all</arg>
    </compilerArgs>
  </configuration>
</plugin>
```

Schritt 7: Statische Codeanalysen (2/2)

- SpotBugs als **<plugin/>** gemäß Anleitung konfigurieren
- Wichtig: **mvn compile** nötig vor der Analyse, da Bytecode analysiert wird
- Analyse mit **mvn spotbugs:spotbugs**
- Ergebnisse ansehen mit **mvn spotbugs:gui**



Schritt 8: Debugging ("In-Vitro-Testing")

- geeignetes Beispiel sollte sich in einer Test Fixture finden lassen
- Achtung: alle aktuellen IDEs verbergen die physische Struktur bei bekannten Typen
- z.B. werden Collections als logische Datenstruktur dargestellt