Lehrstuhl für Software and Computational Systems
Prof. Dr. Gidon Ernst, Thomas Bunk

Übungen zur Vorlesung
Formale Spezifikation und Verifikation

Wintersemester 2021

Übungsblatt 00

Bekanntgabe am 25.10.2021

1 Einrichten einer Java-Entwicklungsumgebung

Im Verlauf des Semesters werden wir zahlreiche Java-Beispiele vorstellen und Programmieraufgaben stellen. In dieser Übungsaufabe wird es daher zunächst darum gehen, ein *Java Development Kit* (JDK) und eine Java IDE einzurichten. Falls Sie zu einer der nachfolgenden Schritte Fragen haben, können Sie diese jederzeit im Zulip Channel dieser Veranstaltung stellen[1].

(a) Falls noch nicht vorhanden, laden Sie sich ein JDK herunter und installieren es. Wir empfehlen eine Version ab Java 11[2]

(b) Laden Sie eine Java IDE Ihrer Wahl herunter und installieren Sie sie. Gängige IDEs sind beispielsweise Eclipse[3] oder IntelliJ IDEA[4]:

(c) Erstellen Sie ein neues Java-Projekt mit folgender Testklasse:

```java
package com.example;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class TestClass {

   @Test
   void test() {
     assertTrue(System.getProperty("java.version").startsWith("11."),
        "Es ist eine andere als die getestete Java-Version installiert");
   }

}
```

Gegebenenfalls müssen Sie anschließend noch die entsprechenden Dependencies zum Projekt hinzufügen. In Eclipse betrifft das beispielsweise die JUnit5 library (siehe Abb. 1).

[1] https://chat.ifi.lmu.de/#narrow/stream/202-SoSy-21W-FSV
[2] https://www.oracle.com/java/technologies/downloads/
[3] https://www.eclipse.org/downloads/packages/
[4] https://www.jetbrains.com/idea/download/

Abbildung 1: Eine Möglichkeit der Eclipse IDE Bibliotheken dem Projekt hinzuzufügen ist via der Option “Add JUnit 5 library to the build path”

(d) Führen sie die Testklasse als JUnit-Test aus und überprüfen Sie, ob der Test erfolgreich war. Falls nicht haben Sie eventuell eine neuere Java-Version in Benutzung. Passen Sie in diesem Fall den Test entsprechend an.

(e) Falls Sie erfolgreich waren, helfen Sie ihren Kommilitonen im Zulip-Chat[1].

2 Testing

Ziel dieser Aufgabe ist das Ausführen einfacher Unit-Tests sowie die Anwendung sogenannter *Property-Based Tests*. Zur Ausführung in Java werden Sie hierfür neben JUnit 5 zusätzlich noch das Java-Plugin `jqwik`[5] benötigen.

Auf Uni2work haben wir außerdem eine Zip-Datei `tests.zip` bereitgestellt. Importieren Sie diese als “Maven-Projekt”, nicht als reguläres Java Projekt. Es beinhaltet ein Java-Projekt einschließlich einer `pom.xml`-Datei für Maven, welche alle notwendigen Dependencies deklariert, darunter

- `org.junit.jupiter 5.8.1`,[6] zur Implementierung und Ausführung von Unit Tests, sowie
- `net.jqwik 1.5.6`[7] zum Ausführen von Property-Based Tests.

Die `pom.xml` enthält außerdem ein Maven Plugin zum nativen Support der oben genannte Test-Funktionalitäten:

- `maven-surefire-plugin 3.0.0-M5`,[8] welches Tests für alle Dateien ended auf `Tests.java`, `Examples.java` oder `Properties.java` ausführt.

Sie können alternativ auch alle Tests von der Kommandozeile heraus ausführen (via `mvn test`), sofern Sie Maven auf Ihrem System installiert haben (sehr leicht auf den meisten Linux Distributionen).

[5] `https://jqwik.net/`
[6] `https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter/5.8.1`
[7] `https://mvnrepository.com/artifact/net.jqwik/jqwik/1.5.6`, frühere Versionen sind ebenfalls ok.
[8] `https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-plugin/3.0.0-M5`

2.1 Unit-Testing

Die folgende Aufgabe bezieht sich auf die Klasse `Simple.java`.

1. Gegeben ist die Methode `int abs(int x)`, die Sie bereits in der Vorlesung kennengelernt haben. Die Methode gibt für einen Integer-Wert `x` ihren absoluten Wert zurück. Schreiben Sie Unit-Tests für diese Methode, beispielsweise in einer Methode `@Test void testAbs()`. Denken Sie dabei über verschiedene Werte zum Testen nach, insbesondere solche, welche interessante Randfälle darstellen könnten.

   Während Sie die Tests implementieren, führen Sie diese außerdem auch als Coverage Tests aus. In Eclipse können Sie dies beispielsweise via "Run" →" Coverage" (`Ctrl+Shift+F11`) erreichen. Dies überprüft, welche Zweige und Codezeilen von `max` tatsächlich durch Unit-Tests abgedeckt werden.

2. Sie finden außerdem die Implementierung einer Methode `int fib(int n)`, welche für einen Wert n den entsprechenden Wert einer Fibonacci-Folge zurückgibt.

   Schreiben Sie Unit Tests für diesen Algorithmus, und überprüfen Sie, wie viel davon durch *Code coverage* abgedeckt ist. Die Tests sollen dabei helfen, Bugs in der Implementierung aufzudecken. Für welche Werte schlägt der Test fehl? Was muss an der Implementierung geändert werden, um dies zu beheben?

3. Gegeben ist außerdem die Methode `int max(int[] x, int left, int right)`, welche für ein gegebenes Array bestehend aus Integern den größten Wert innerhalb der angegebenen Bereiche zurückliefert.

   Schreiben Sie Unit-Tests für diese Methode. Welches sind die hierfür zu testenden Randfälle? Denken Sie dabei über verschiedene Kombinationen von Test-Werten nach, einschließlich auch solche, welche die interessanten Randfälle darstellen.

2.2 Property-Based Testing

In dieser Teilaufgabe geht es nun darum, die oben genannten Methoden mittels *Property-Based Testing* und dem `jqwik`-Framework zu überprüfen. Formulieren Sie die jeweiligen Bedingungen als eine oder mehrere `@Property` Methoden, und führen Sie diese als JUnit Test-cases aus, um den Code zu überprüfen.

1. Die Methode `int abs(int x)` kann man so charakterisieren:
   - Der Rückgabewert ist entweder `x` oder `-x`
   - Der Rückgabewert ist nicht negativ
2. Für `int max(int[] x, int left, int right)` haben wir ebenfalls zwei Korrektheitskriterien die von Interesse sind:
   - der zurückgegebene Wert sollte im Parameter `x` enthalten sein
   - der zurückgegebene Wert sollte mindestens so groß wie alle Werte von `x` sein

   ~~Hierzu können Sie z.B. die Java-API verwenden, um einfach zu überprüfen, ob ein Wert im Array ist: `List.of(x).contains(_)`.~~ Leider funktioniert dies nicht für Arrays von `int`, Sie müssen diesen Code also selber schreiben.

2.3 Einige weitere Properties

Nachfolgendend sollen einige einfache Properties von `Strings` mittels Properties überprüft werden. In jedem der untenstehenden Fälle ist die linke Seite einer Gleichung gegeben. Finden Sie die entsprechende rechte Seite und stellen Sie die Korrektheit sicher mittels dem Erstellen entsprechender `jqwik` properties.

Variablen `x,y` sind vom Typ `String`, Variablen `c` sind vom Typ `char` und Variablen `i` sind vom Typ `int`. Ergänzen Sie die fehlenden Abschnitte `_`.

Nachfolgend die Auswirkung der Konkatenierung auf die Länge:

- `(x + "").length() == _`
- `("" + x).length() == _`
- `(x + x).length() == _`
- `(x + y).length() == _`

Wie verhält sich `charAt` bezüglich der Konkattenierung? Ergänzen Sie die Fallunterscheidung auf der rechten Seite der Gleichung:

- `(x + y).charAt(i) == (_?_:_)`

Unter welchen Bedingungen ist das wohldefiniert, d.h. wirft keine Exception?

- Sie können diese Bedingung mittels `Assumptions.assumeTrue` am Anfang des Tests ausdrücken, jqwik führt den Test nur für solche Zufallswerte aus, die die Annahme erfüllen
- Sie können alternativ eine entsprechende `if`- Bedingung um den Body der Test-Methode platzieren. Was ist der Nachteil dieses Ansatzes?

Die `String`-Klasse bietet viele weitere Methoden, welche jeweils in Relation zu den Anderen gestellt werden kann, wie beispielsweise `trim`, `replace`, `contains`, `indexOf`. Nicht alle interessanten Properties können als Gleichungen ausgedrückt werden, z.B.:

- was können Sie über `x.trim().length()` sagen?