Informatik II (D-ITET)
Frühjahrssemester 2017
Prof. Friedemann Mattern
Hossein Shafagh (shafagh@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Gruppe für Verteilte Systeme
http://vs.inf.ethz.ch

Übungsserie Nr. 0

Ausgabe: 22. Februar 2017
Keine Abgabe
Keine Bewertung

Übungen

Bitte stellen Sie sicher, dass Sie bei mystudies für die Vorlesung eingeschrieben sind. Sie erhalten dann eine Nachricht mit weiteren Informationen zur Einschreibung in eine Übungsgruppe. Besuchen Sie ausserdem regelmässig die Webseite der Vorlesung: http://www.vs.inf.ethz.ch/edu/I2/ – Dort finden Sie regelmässig aktuelle Informationen zur Vorlesung, den Übungen und der Prüfung. Beachten Sie auch die Informationen zum *Einführungskurs in Java* sowie zu *Programmieren in Java*.

Unterlagen

Für diese Serie benötigen Sie das Archiv

http://vs.inf.ethz.ch/edu/FS2017/I2/downloads/u0.zip

1. Aufgabe: HelloWorld.java

ETH Codeboard link: https://codeboard.ethz.ch/ifee2u0a1

(1a) Falls Sie die Übungen auf Ihrem privaten System bearbeiten wollen, installieren Sie das Java Development Kit (JDK) für Java SE 8[1,2,3].

(1b) Entpacken Sie das Archiv dieser Serie und bringen Sie das darin enthaltene Programm *HelloWorld* auf der Kommandozeile zur Ausführung. Führen Sie dazu im entsprechenden Verzeichnis folgende Schritte aus:

[1] http://docs.oracle.com/javase/8/docs/
[2] http://www.oracle.com/technetwork/java/javase/install-142943.html
[3] http://www.oracle.com/technetwork/java/javase/downloads/index.html

```
$ javac u0a1/HelloWorld.java
$ java u0a1.HelloWorld
```

Der erste Schritt kompiliert das Programm und erzeugt die Datei *HelloWorld.class*. Diese enthält das übersetzte Programm in ausführbarer Form, in sogenanntem *Bytecode*. Mit dem zweiten Schritt wird die *Java Virtual Machine* gestartet und das Programm ausgeführt, indem die Methode `main` der Klasse `HelloWorld` aufgerufen wird. Beachten Sie, dass die Methode `main` als `public` und `static` deklariert sein muss. Beachten Sie ausserdem, dass die Klasse im `package` *u0a1* liegt. Das spiegelt sich sowohl durch das gleichnamige Verzeichnis als auch durch die entsprechende Deklaration in *HelloWorld.java* wieder.

(1c) Obwohl ein Texteditor und die oben genannten Programme hinreichend für die Entwicklung von Java-Programmen sind, bietet der Einsatz einer integrierten Entwicklungsumgebung (IDE, Integrated Development Environment) viele Vorteile. In der Java-Welt gibt es z.B. *Eclipse, IntelliJIDEA, BlueJ, JBuilder, NetBeans* oder *JCreator*. Sie können frei wählen, in welcher Umgebung Sie die Übungen bearbeiten wollen. Beachten Sie allerdings, dass Ihnen Ihr Tutor *nur für Eclipse*[4]) Hilfe anbieten kann.
Machen Sie sich mit Ihrer Entwicklungsumgebung vertraut. Nutzen Sie dazu auch die zahlreichen Anleitungen und Foren, die es im Internet gibt. Erstellen Sie ein Projekt für *Hello World* und bringen Sie es in Ihrer Entwicklungsumgebung zur Ausführung.

(1d) Das Programm ist auch auf Codebord verfügbar. Führen Sie es aus und submitten Sie es um ihre Codeboard-Umgebung zu testen.

2. Aufgabe: Das erste Java Program

ETH Codeboard link: `https://codeboard.ethz.ch/ifee2u0a2`

(2a) Bringen Sie das im Archiv dieser Serie enthaltente Programm `u0a2.Main` in Ihrer Entwicklungsumgebung zur Ausführung.

(2b) Das Programm besteht aus zwei Teilen, einer Klasse `Main` und einer Klasse `Signum`. Beachten Sie, dass die *.java* Dateien so heissen müssen wie die jeweils darin enthaltene Klasse. Die Methode `signum` aus der Klasse `Signum` implementiert die *Signum*-Funktion

$$sgn(x) := \begin{cases} -1, & \text{if } x < 0 \\ 0, & \text{if } x = 0 \\ +1, & \text{if } x > 0 \end{cases}$$

Die Methode `main` aus der Klasse `Main` ruft `signum` für verschiede Werte auf und gibt die Ergebnisse auf der Standardausgabe aus. Dadurch kann man überprüfen, ob die Implementierung von `signum` korrekt ist.
Ergänzen Sie die `main` um einen Aufruf von `signum` mit einer positiven Zahl und überprüfen Sie die Ausgabe Ihres Programms.

[4]Empfehlung: Paket *Eclipse IDE for Java Developers* von: `www.eclipse.org/downloads`

(2c) Das Programm ist auch auf Codebord verfügbar. Fügen Sie Ihre Lösung ein und submitten Sie sie.

3. Aufgabe: Automatisiertes Testen

ETH Codeboard link: `https://codeboard.ethz.ch/ifee2u0a3`

(3a) In der letzten Aufgabe haben Sie visuell überprüft, ob die Ergebnisse der `signum` Funktion korrekt sind. Dieser manuelle Ansatz ist bei grösseren Programmen mit vielen Tests allerdings sehr aufwändig und fehleranfällig. Daher ist eine automatisierte Verifikation von Testergebnissen zu bevorzugen. JUnit4[5] ist eine Java-Bibliothek für diesen Zweck. Installieren Sie JUnit4 auf Ihrem System und machen Sie sich mit der JUnit4 Bibliothek vertraut[6].

(3b) Werfen Sie einen Blick auf die Klasse `Tests`. Führen Sie dann die Tests in der Kommandozeile aus (also ausserhalb ihrer IDE), indem Sie die folgenden Befehle eingeben:

```
$ export CLASSPATH=/usr/share/java/junit4.jar:.
$ javac u0a3/*.java
$ java org.junit.runner.JUnitCore u0a3.Tests
```

Beachten Sie, dass Sie sich dazu in demjenigen Verzeichnis befinden müssen, in welchem das Verzeichnis *u0a3* liegt. Ausserdem müssen Sie gegebenenfalls den Pfad zur *junit4.jar*, der JUnit4 Bibliothek, entsprechend der Installation auf Ihrem System ändern (z.B. auf `junit-4.11.jar` oder Ähnliches).

Hinweis: unter Windows werden die Verzeichnisse im *CLASSPATH* durch Semikolon anstatt durch Doppelpunkt getrennt.

(3c) Finden Sie durch Internetsuche heraus, wie man die Tests in Ihrer Umgebung ausführen kann. Hinweis: Eclipse bringt die JUnit-Bibliothek bereits mit. Achten Sie allerdings darauf, dass Sie die *Version 4* von JUnit verwenden.

(3d) Ergänzen Sie die Tests um einen Fall mit einer positiven Zahl und führen Sie die Tests in Ihrer Entwicklungsumgebung aus.

(3e) Verändern Sie die Implementierung von `signum` so, dass manche Tests fehlschlagen. Führen Sie die Tests in Ihrer Entwicklungsumgebung aus und analysieren Sie die Ausgaben. Machen Sie sich damit vertraut, wie Sie innerhalb Ihrer Entwicklungsumgebung schnell von den Ausgaben fehlgeschlagener Tests zu deren Ursache gelangen können.

[5] Download auf `http://junit.org/`

[6] Empfehlung: Tutorial auf `http://www.cavdar.net/2008/07/21/junit-4-in-60-seconds/` oder das erweiterte Tutorial auf `http://www.vogella.com/articles/JUnit/article.html`

4. Aufgabe: Gerichtete Graphen

ETH Codeboard link: `https://codeboard.ethz.ch/ifee2u0a4`

Ein gerichteter Graph besteht aus einer Menge von Kanten und einer Menge von Knoten, wobei jede Kante als Pfeil dargestellt wird, welcher einen Ausgangsknoten mit einem Zielknoten verbindet. Knoten stellt man im Allgemeinen in Form von Kreisen dar. Betrachten Sie die folgende Problemstellung:

> Sie besitzen drei Kannen; die erste fasst 8 Liter und ist bis zum Rand mit Wasser gefüllt, die zweite fasst 5 Liter und ist leer, die dritte fasst 2 Liter und ist ebenfalls leer. Dies ist der Startzustand. Eine Umschüttung von Wasser aus einer Kanne in eine andere führt von einem Zustand in einen anderen, wobei entweder eine Kanne vollständig geleert oder vollständig gefüllt werden muss. Ein Zustand wird durch das Tripel (a, b, c) kodiert, wobei a der Inhalt der 8-Liter-Kanne, b der Inhalt der 5-Liter-Kanne und c der Inhalt der 2-Liter-Kanne ist.

(4a) Zeichnen Sie den Graphen, der aus allen möglichen Zuständen (= Knoten) und möglichen Umschüttungen (= gerichteten Kanten) besteht. Markieren Sie die Knoten des Graphen mit den Zuständen, die durch die Folge der Umschüttungen auftreten. Achten Sie darauf, dass Knoten mit gleichem Tripel nicht mehrfach im Graphen auftreten.

(4b) Versehen Sie die Knoten des Graphen mit der minimalen Anzahl von benötigten Umschüttungen, um den entsprechenden Zustand vom Startzustand ausgehend zu erreichen. Wie gehen Sie dabei vor?

(4c) Ausgehend vom Startzustand, was ist die maximale Anzahl von Umschüttungen die benötigt wird, um einen beliebigen Zustand zu erreichen? Was ist die mittlere Anzahl?