Softwareentwicklung I (IB)

Blatt 6

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 21.12.2017 09:52

Vorbemerkung zum Zulassungsschein

Die Abgabe dieses Blattes ist nicht mehr erforderlich für den Zulassungsschein. Dennoch üben Sie auf diesem Blatt wichtige Dinge. Die Objektorientierung ist Gegenstand der Klausur. Außerdem werden Sie hier an das Arbeiten mit einer IDE herangeführt. Dies ist notwendig für Softwareentwicklung II (IB) im nächsten Semester.

Wenn Sie Feedback für die bearbeiteten Aufgaben auf diesem Blatt haben möchten, erzeugen Sie ein entsprechendes Issue und weisen Sie es den Tutoren und mir zu.

IntelliJ IDEA

Neben Editoren wie Atom gibt es auch sogenannte IDEs. IDE steht für Integrated Development Environment. Populär für Java sind Eclipse, Netbeans und IntelliJ IDEA.

Sie können nach wie vor mit dem Atom entwicklen, Sie können aber auch eine der IDEs nutzen. Ich selbst nutze IntelliJ IDEA und werde Ihnen dazu im Folgenden die notwendigen Schritte erläutern um ab sofort mit IntelliJ IDEA zu entwickeln.

Auf den Laborrechnern

Auf den Laborrechnern ist IntelliJ IDEA bereits installiert. Wenn Sie es zum ersten Mal starten, müssen Sie den Lizenzserver-Punkt auswählen. Der Server selbst sollte dann gefunden werden. Anschließend können Sie damit arbeiten.

Auf dem eigenen Rechner

Sie können sich IntelliJ IDEA in zwei verschiedenen Versionen herunterladen. Die Community-Edition sollte für diese Veranstaltung eigentlich reichen. Als Studierender können Sie aber auch die Ultimate-Edition kostenlos nutzen. Mehr Informationen dazu finden Sie unter https://www.jetbrains.com/student/.

IntelliJ IDEA konfigurieren

Grundsätzlich können Sie IntelliJ IDEA einfach nutzen, aber IntelliJ IDEA rückt beispielsweise immer 4 Leerzeichen ein und das gefällt unserer Checkstyle-Konfiguration nicht.

Google Java Style

Laden Sie die Datei https://github.com/google/styleguide/blob/gh-pages/intellij-java-google-style.xml herunter. Gehen Sie anschließend auf `Preferences` → `Editor`→`Code Style`→`Java` und wählen Sie bei dem Zahnrad `Import Scheme` → `IntelliJ IDEA Code Style XML`. Öffnen Sie die gerade herunter geladene Datei und importieren Sie sie. Wählen Sie das `Scheme GoogleStyle` aus und klicken Sie `OK`.

Installieren Sie außerdem das `google-java-format`-Plugin und **aktivieren** Sie es.

Unter `Preferences`→`Plugins` klicken Sie auf `Browser Repositories`. Dort können Sie in dem Suchfeld `google-java-format` eingeben und es installieren.

Unter `Preferences`→`Other Settings` können Sie es für das aktuelle Projekt aktivieren.

Jetzt können Sie mit `Code`→`Reformat Code` bzw. dem dort zu findenden Tastaturkürzel den Code automatisch richtig formatieren lassen!

Unter `File`→`Other Settings`→`Default Settings` finden Sie die `google-java-format`-Settings für alle zukünftigen Projekte als Grundeinstellung.

Checkstyle

Sie können Checkstyle über die Gradle-Integration aufrufen und bekommen die Fehler zum Lesen auf der eingebetteten Console oder Sie installieren das Checkstyle IDEA-Plugin.

Wenn Sie es gefunden und installiert haben, finden Sie die Einstellungen für das aktuelle Projekt und als Default genauso wie vorher für den Google Code Style. Dort können Sie direkt die Google Checks auswählen.

Sie finden dann unten im IntelliJ IDEA-Fenster einen Reiter `Checkstyle`. Wenn Sie diesen öffnen, können Sie Ihre Sourcen mit Checkstyle überprüfen.

Aufgabe 1

Das Repository für diese Aufgabe bekommen Sie unter https://classroom.github.com/a/7JOQZsdf.

Umgang mit IntelliJ IDEA

Sie können das Projekt direkt mit IntelliJ IDEA clonen und importieren. Es ist dann automatisch mit GitHub verknüpft und Sie können aus IntelliJ IDEA heraus auch committen und pushen.

Wenn Sie das Projekt ausgewählt haben, sollte sich ein Fenster `Import Project from Gradle` öffnen. Alles was defaultmäßig aus- oder abgewählt ist, sollte für unsere Zwecke passen. Klicken Sie auf `OK`. IntelliJ IDEA braucht jetzt eine Weile um das Projekt zu importieren.

Nachdem es fertig ist, sehen Sie unten ein Fenster mit den Build-Ausgaben. Links sehen Sie einen Reiter `Project`. Wenn er noch nicht geöffnet ist, klicken Sie darauf und Sie sehen Ihre Dateien und Verzeichnisse. Neue Dateien legen Sie mit Rechtsklick auf die entsprechenden Verzeichnisse an.

Rechts sehen Sie einen Reiter `Gradle`. Wenn Sie darauf klicken, öffnet sich die Gradle-Konfiguration. Mit einem Klick auf den Button mit dem runden, grünen Gradle-Logo können Sie sog. Run-Configurations für Gradle-Tasks erzeugen. Klicken Sie darauf und tragen Sie bei `Command line` z.B. `compileJava` ein. Nachdem Klicken von `OK` wird `build` sofort ausgeführt. Über dem Gradle-Fenster sehen Sie den Namen der Run-Configuration und jedesmal wenn Sie den grünen Play-Button drücken wird das wieder ausgeführt.

Eine weitere hilfreiche Run-Configuration die Sie so erzeugen können, hat die Command line `test`. Damit können Sie die Tests ausführen, die im Moment noch nicht mal compiliert werden können, weil Ihre Klasse (siehe unten) noch fehlt. Später haben Sie auch eine schöne Darstellung der Tests die fehlgeschlagen.

Eine Run-Configuration für Checkstyle brauchen Sie nur dann, wenn Sie das Checkstyle-Plugin nicht verwenden (wollen).

Die für diese Aufgabe noch interessante Run-Configuration ist die mit der Command line `run -q`. Damit wird Ihr Programm ausgeführt. Das Programm ist dieses Mal interaktiv, d.h. Sie können etwas eingeben während es läuft (siehe unten). `-q` steht für `quiet` und ist dabei sehr sinnvoll, sonst gibt Gradle zwischen Ihren Eingaben und den Ausgaben auch immer irgendwelche Fortschrittsmeldungen aus.

Zwischen den verschiedenen Run-Configurations können Sie dann neben dem Play-Button wählen und sie mit Play ausführen.

Unter `VCS→Git` finden Sie alles was Sie zum Committen und Pushen brauchen. VCS steht übrigens für Version Control System.

Das Vorgehen für IntelliJ IDEA ist bei allen weiteren Projekten genauso.

Und jetzt die eigentliche Aufgabe

Schreiben Sie ein Klasse `Counter` die einen Zähler verwaltet. Wird ein Objekt dieser Klasse erzeugt, repräsentiert es zunächst den Wert 0 des Zählers. Mit einer Methode kann der Zähler um jeweils 1 inkrementiert werden. Mit einer zweiten Methode wird der Zähler auf 0 zurück gesetzt.

Sie finden in Ihrem GitHub-Repository eine Klasse `CounterApp`, die die Klasse `Counter` nutzt. Versuchen Sie den Code der Klasse `CounterApp` zu verstehen. In dieser Klasse erkennen Sie auch wie der Zugriff auf den Wert des Zählers sowie die beiden Methoden heißen müssen. **Verändern Sie die Klasse `CounterApp` nicht!**

Eine Beispielsitzung von `CounterApp` könnte folgendermaßen aussehen:

```
counter = 0
add
counter = 1
> add
counter = 2
> +
counter = 3
> +
counter = 4
> add
counter = 5
> reset
counter = 0
> ad
unknown command: ad
counter = 0
> +1
unknown command: +1
counter = 0
> hallo
unknown command: hallo
counter = 0
> +
counter = 1
> 0
counter = 0
> quit
counter = 0
Bye-bye!
```

Aufgabe 2

Das Repository für diese Aufgabe bekommen Sie unter https://classroom.github.com/a/0-h_MVk0.

Ihr Kunde, eine große Kaufhauskette, plant eine Payback-Card einzuführen. Über die Kunden werden bereits Datensätze in einer Datenbank gespeichert, die genutzt werden können. Auf der Kundenkarte soll daher nur folgendes gespeichert werden:

- Kundennummer
- Name des Kunden
- die aktuelle Anzahl seiner Treuepunkte (bonus points)
- Premiumkunde oder nicht

Implementieren Sie die Klasse `PaybackCard`. Verwenden Sie als Bezeichner für die Objektvariablen genau folgende: `customerid`, `name`, `bonusPoints` und `premiumCustomer`.

Implementieren Sie außerdem die Methode `print`. Diese soll folgendermaßen funktionieren:

- Wenn der `name` den Wert `null` hat, soll ausgegeben werden:

  ```
  Kundenname fehlt!
  ```

- Wenn der `name` zwar einen sinnvollen Wert hat, aber die Kundennummer noch `0` ist, soll ausgegeben werden:

  ```
  Kundennummer fehlt!
  ```

- Sonst soll folgendes ausgegeben werden:

  ```
  <name> (Kundennummer: <kundennummer>), <anzahl> Punkte, <premium>
  ```

  mit folgenden Besonderheiten:

  - wenn die Anzahl von Bonuspunkten 0 ist, soll stattdessen `keine` ausgegeben werden
  - wenn der Kunde Premiumkunde ist, soll `Premiumkunde` ausgegeben werden,
  - wenn nicht, soll nichts ausgegeben werden
  - im letzten Fall, soll auch das Komma nach `Punkte` nicht ausgegeben werden

- Beispiele

  ```
  Jesus Lizard (Kundennummer: 815), keine Punkte
  Steve Albini (Kundennummer: 1234), 1000 Punkte, Premiumkunde
  ```

Außerdem soll eine Payback-Karte mit der Methode `printToJson` als JSON ausgegeben werden können. JSON ist die **J**ava**S**cript **O**bject **N**otation und wird von vielen Web-Diensten als Austauschformat genutzt. Die beiden o.a. Payback-Karten sehen in JSON wie folgt aus:

```
{
  "name": "Jesus Lizard",
  "customerid": 815,
  "bonusPoints": 0,
  "premiumCustomer": false
}
```

bzw.

```
{
  "name": "Steve Albini",
  "customerid": 1234,
  "bonusPoints": 1000,
  "premiumCustomer": true
}
```

Eine Karte ohne Namen und Kundennummer kann auch in JSON ausgegeben werden:

```
{
  "name": null,
  "customerid": 0,
  "bonusPoints": 0,
  "premiumCustomer": false
}
```

Ein JSON-Objekt steht also immer in geschweiften Klammern. Die Objektvariablen stehen mit ihren Werten als Paare durch den Doppelpunkt getrennt. Mehrere Objektvariablen werden durch Kommata getrennt. Die Objektvariablen sind in JSON Strings, werden also in Anführungszeichen ausgegeben. Die Werte werden in dem Fall wie in Java angegeben, auch das `null`. Geben Sie das JSON-Objekt im angegebenen Layout aus.

Implementieren Sie in der Klasse `PaybackCardApp` das dazugehörige ausführbare Programm. Der Ablauf soll ähnlich wie in der Aufgabe 1 sein. Wenn Sie das Programm starten, werden Sie mit folgender Zeile aufgefordert ein Kommando einzugeben:

```
Geben Sie ein Kommando ein:
```

Reagieren Sie auf folgende Eingaben, wie jeweils beschrieben:

h / ? / help gibt folgende Hilfe aus:

```
h / help / ?     --- gibt diese Info aus
n / name         --- setze Namen
k / kundennummer --- setze Kundennummer
b / bonuspunkte  --- addiere Bonuspunkte
p / premiumkunde --- setze Premium-Status
d / drucke       --- gebe Karteninfo aus
j / json         --- gebe Karteninfo als JSON aus
q / quit         --- beenden
```

d / drucke gibt die Karte mit Hilfe der Methode print aus, z.B.

```
Geben Sie ein Kommando ein:
d
Jesus Lizard (Kundennummer: 1234), 1100 Punkte
Geben Sie ein Kommando ein:
d
Kundenname fehlt!
Geben Sie ein Kommando ein:
d
Kundennummer fehlt!
```

j / json gibt die Karte als JSON aus

n / name setzt den Namen. Hier muss zunächst noch nach dem Namen gefragt werden, die Sitzung sieht also z.B. so aus:

```
Geben Sie ein Kommando ein:
n
Geben Sie den Namen ein:
Jon Spencer
```

Dabei sind `n` und `Jon Spencer` die Eingaben, die der Nutzer tippen muss, alles andere wird vom Programm ausgegeben.

k / kundennummer setzt die Kundennummer. Hier wird auch noch nach der Kundennummer gefragt:

```
Geben Sie ein Kommando ein:
k
Geben Sie die Kundennummer ein:
1234
```

Achtung: Die eingegeben Zahl ist ein `String`. Sie können diesen aber wie gewohnt in einen `int` umwandeln.

b / bonuspunkte addiert die eingegebenen Bonuspunkte zum bisherigen Stand. Hier muss auch noch nach der Anzahl gefragt werden. Anschließend wird der aktuelle Stand ausgegeben:

```
Geben Sie ein Kommando ein:
b
Geben Sie die Bonuspunkte ein:
234
Neuer Stand: 1234 Punkte.
```

p / premiumkunde setzt den Premiumstatus. Dazu muss noch gefragt werden ob der Premiumstatus gesetzt werden soll. Wenn `j` oder `J` eingegeben wird, soll er gesetzt werden, sonst nicht.

```
Geben Sie ein Kommando ein:
p
Ist der Kunde Premiumkunde? (j/N)
j
```

Es ist üblich mit einem Großbuchstaben (in dem Fall das `N` bei `(j/N)`) anzuzeigen was übernommen wird, wenn irgendetwas ungültiges eingegeben wird.

q / quit beendet das Programm und verabschiedet sich vorher mit `Bye-bye!`.