technische universität
dortmund

Bearbeitet von:
Sebastian Hauer
Sergej Karsten

Fakultät für Informatik
Lehrstuhl 1
Dr. Lars Hildebrand
Iman Kamehkhosh, Marcel Preuß, Henning Timm,
Alexander Lochmann, Bartho Rudak, Thomas Schmitz

Praktikum zu

Einführung in die Informatik für LogWings und WiMas

Wintersemester 2014/15

Übungsblatt 3

Bearbeitungszeit:
10.11.14 -
14.11.14

Aufgabe 3.1 – Blatt 2 vervollständigen

Besprechen Sie mit ihrer Praktikumsgruppe das zweite Übungsblatt zuende.

Aufgabe 3.2 – Fehlersuche per Hand

In dieser Aufgabe sollen Sie nicht programmieren. Bearbeiten Sie stattdessen die Aufgabe nur auf dem Papier – **so, wie Sie es auch in der Klausur tun müssen**.

- Betrachten Sie das untenstehende Java-Programm. Das Programm enthält **Syntaxfehler**, die dazu führen, dass das Programm nicht kompiliert werden kann. Unterstreichen Sie die Syntaxfehler im Programmtext und korrigieren Sie sie auf den entsprechenden Linien daneben.
- Werden **Laufzeitfehler** auftreten, die dazu sorgen werden, dass das Programm zwar kompiliert, aber während der Ausführung abstürzt, wenn das Programm nach der Korrektur der Syntaxfehler ausgeführt wird? Wenn ja, welche Laufzeitfehler erwarten Sie?

- Nachdem Sie die **Syntaxfehler** behoben haben und den einzelnen semantischen Fehler korrigiert haben, der den Laufzeitfehler verursacht, was wird das Programm Ihrer Meinung nach ausgeben?

- Enthält das Programm außerdem **Semantikfehler**, die dazu führen, dass das Programm nicht das berechnet, was von ihm erwartet wird? Wenn ja: Wie müssen die Fehler korrigiert werden:

Der Programmtext:

```
1  Class WochenImSchaltjahr {
2    public static void main (string[] args) {
3      integer tage;
4      int tageprowoche = 0;
5      int wochen;
6      int resttage;
7
8      tage = 366
9      wochen = tage * tageprowoche;
10     resttage = tage % tageprowoche;
11
12     System.out.pint("Ein Schaltjahr besteht");
13     //System.out.print(" manchmal");
14     Systemout.println(" aus " + wochen +
15     " Wochen und "  resttage + " Tagen.);
16   {
17 }
```

Aufgabe 3.3 – Deklaration, Initialisierung und Zuweisung

Schauen Sie sich folgendes Programm an und überlegen Sie, was die Ausgabe sein könnte.

```
1  class DeklarationBeispiel {
2      public static void main(String[] args) {
3          int a;
4          a = 7;
5          int b = 5;
6          System.out.println("a: " + a + ", b: " + b);
7          int c = a + 3;
8          int d;
9          System.out.println("c: " + c);
10         a = b;
11         System.out.println("a: " + a + ", c: " + c);
12     }
13 }
```

Worin unterscheiden sich die Zeilen 3 und 4 von der Zeile 5?

Notieren Sie die von Ihnen vermutete Ausgabe:

Führen Sie das Programm nun in Java aus (nachdem Sie es abgeschrieben haben) und überprüfen Sie Ihre Vermutung. Stimmt die Ausgabe mit ihrer Vermutung überein? Begründen Sie.

Aufgabe 3.4 – Debugging

Debugging nennt man den Prozess des Suchens von Fehlern in einem Programm. Eine der gewöhnlichsten Formen des Debuggings ist das sogenannte „print“-Debugging, bei welchem Sie, wie in Aufgabe 3.3, Werte von Variablen zu kritischen Momenten ausgeben und mit ihren vermuteten Ergebnissen vergleichen. Eine wesentlich mächtigere Form des Debuggings ist das Verwenden eines sogenannten „Debuggers“, welcher es ermöglicht den Programmablauf zu einer gewählten Stelle im Quellcode zu unterbrechen und „in das Programm hinein zu sehen“.

Um den Programmfluss zu einem bestimmten Moment zu unterbrechen, werden sogenannte „Breakpoints“ verwendet. Ein Breakpoint kann in eine Zeile im Quellcode gesetzt werden, vor deren Ausführung der Debugger den Programmfluss anhalten wird. Einen Breakpoint können Sie in eine Zeile setzen, indem Sie im Quelltexteditor links auf den grau hinterlegten Rand der Zeile doppel-klicken, oder rechts-klicken und `Toogle Breakpoint` auswählen. Auf diese Weise sollten Sie übrigens auch die Zeilennummerierung in Eclipse anschalten, falls Sie das noch nicht getan haben.

Der Eclipse-Debugger kann dann durch das Klicken auf den kleinen, grünen Käfer neben dem Run-Button gestartet werden.

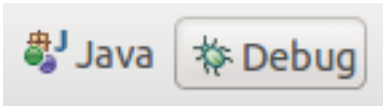
Wenn Sie das Programm über den Debugger starten, wird Eclipse seine Ansicht wechseln wollen, sobald ein Breakpoint erreicht wird. Akzeptieren Sie diesen Vorschlag. Ihr Quellcode wird in den unteren Teil des Bildschirms verschoben und in der oberen Hälfte des Bildschirms erscheinen Werkzeuge zum Analysieren Ihres Programmes. Dabei ist für das Praktikum nur die Tabelle im oberen rechten Fenster relevant. In dieser werden alle im Programmkontext initialisierten Variablen und deren Werte angezeigt.

Im oberen Bereich ihrer Arbeitsumgebung können Sie durch das Klicken des grünen Pfeiles in Form eines „Abspielen“-Symboles den Programmfluss bis zum Eerreichen des nächsten Breakpoints fortsetzen, oder durch Klicken des sich rechts daneben befindenden Buttons mit dem Tooltip `Step Over(F6)` das Programm genau eine Anweisung durchführen lassen, um so schrittweise durch das Programm zu wandern (im Fachjargon daher auch „steppen“ genannt).

Debuggen Sie das Programm aus Aufgabe 3.3, indem Sie einen Breakpoint in die zweite Zeile der `main`-Methode (`a = 7;`) setzen und notieren Sie anschließend den Inhalt der Variablenbelegungen nach jedem Schritt.

(Sie können in die erste Zeile dieses Programmes keinen Breakpoint setzen.)

Sie können die Debug-Ansicht wieder verlassen, indem Sie in der oberen rechten Ecke des Bildschirms auf die Java-Ansicht wechseln.



Aufgabe 3.5 – Wahrheitswerte

Der Datentyp `boolean` dient zur Speicherung von Wahrheitswerten und ist dadurch zur Beantwortung von „Ja/Nein"-Fragen zu verwenden. Im nächsten Übungsblatt werden wir mit Hilfe von `booleans` den Programmfluss kontrollieren und zur Vorbereitung dessen beschäftigen wir uns nun mit der Auswertung von Ausdrücken zu bool'schen Werten.

Erstellen Sie eine neue Klasse und übernehmen Sie folgendes Programm:

```
1  class BooleanBeispiel {
2      public static void main(String[] args) {
3          int a = 5;
4          int b = 2;
5          boolean t = a > b;
6          boolean f = a > 7;
7          a = 10;
8          System.out.println("a: " + a + " , b: " + b);
9          System.out.println("t: " + t + " , f: " + f);
10         boolean and = t && f;
11         boolean or = t || f;
12         boolean complex = true && ((a < 4) || b > 1);
13         System.out.println("and: " + and + " , or: "
14             + or + " , complex: " + complex);
15     }
16 }
```

Debuggen Sie das Programm wie das Vorherige, indem Sie diesmal einen Breakpoint in die erste Zeile des Programms setzen. Notieren Sie schrittweise die Auswertung der Zuweisung der Variablen `complex` per Hand (der Debugger hilft Ihnen hier *nicht*, da er die Zuweisung in einem Schritt ausführt).

Zum Vergleich, werten Sie den Ausdruck `a < 0 && b < 0` in so wenig — begründeten — Schritten wie möglich aus:
