

SE1, Aufgabenblatt 7

Softwareentwicklung I – Wintersemester 2016/17

Schleifen

Moodle-URL: uhh.de/se1
Feedback zum Übungsblatt: uhh.de/se1-feedback
Projektraum Softwareentwicklung 1- WiSe 2016/17
Ausgabewoche 01. Dezember 2016

Kernbegriffe

Neben den Kontrollstrukturen *Sequenz* und *Auswahl* gibt es die *Wiederholung* (engl.: repetition). Diese wird in Java überwiegend durch *Schleifenkonstrukte* (engl.: loop constructs) realisiert, die ermöglichen, dass eine Reihe von Anweisungen mehrfach, nur einmal oder gar nicht ausgeführt wird. Grundlegend lassen sich u.a. *Zählschleifen* (in Java mit `for` möglich) und *bedingte Schleifen* (in Java mit `while` und `do-while` möglich) unterscheiden. Die Wiederholung durch Schleifenkonstrukte wird auch *Iteration* (engl.: iteration) genannt.

Schleifenkonstrukte bestehen immer aus zwei Teilen: dem *Schleifenrumpf* (engl.: loop body), der die zu wiederholenden Anweisungen enthält, und der *Schleifensteuerung* (loop control), die die Anzahl der Wiederholungen bestimmt. Eine Schleife ist *abweisend* oder *kopfgesteuert*, wenn es dazu kommen kann, dass der Schleifenrumpf gar nicht ausgeführt wird; wird der Schleifenrumpf auf jeden Fall mindestens einmal ausgeführt, ist die Schleife *nicht-abweisend* oder *endgesteuert*. Hängt die (jeweils nächste) Ausführung des Schleifenrumpfes von einer Bedingung ab (in Java bei allen Schleifen der Fall), kann die Schleife *positiv bedingt* sein („Rumpf ausführen, *solange* die Bedingung zutrifft“) oder *zielorientiert bedingt* („ausführen, *bis* die Bedingung zutrifft“). In Java sind alle Schleifen positiv bedingt.

Jeder Variablen (Exemplarvariable, formaler Parameter, lokale Variable) in einem Java-Programm ist ein *Sichtbarkeitsbereich* (engl.: scope) zugeordnet, in dem sie im Quelltext benutzt werden kann. Er entspricht der Programmeinheit, in der die Variable deklariert ist.

Die *Lebensdauer* (engl.: lifetime) einer Variablen oder eines Objektes ist die Zeit *während der Ausführung eines Programms*, in der der Variablen oder dem Objekt Speicher zugeteilt ist. Lokale Variablen werden beispielsweise auf dem Aufruf-Stack abgelegt und nach Ausführung ihrer Methode automatisch wieder entfernt, sie leben also maximal für die Dauer einer Methodenausführung. Die Lebensdauer von Referenzvariablen muss nicht gleich der Lebensdauer der referenzierten Objekte sein.

Lernziele

Programmgesteuerte wiederholte Ausführung verstehen; Unterschiede zwischen den verschiedenen Schleifenkonstrukten kennen; Debugger einsetzen.

Aufgabe 7.0


 Tragt auf der Rückseite des Abnahmezettels eure Daten von der Vorderseite ein, und zwar in folgendem Format:

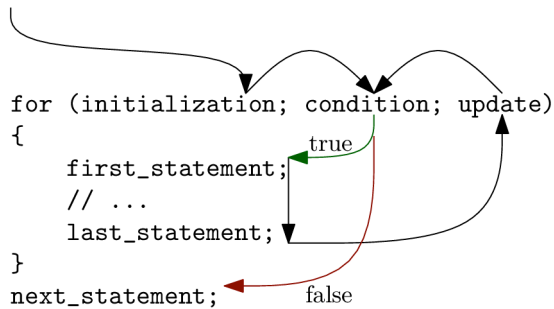
Mustermann, Max 1234567 Divo Physik

Aufgabe 7.1 Vorgegebene Schleifen analysieren

7.1.1 Öffnet das Projekt *Iteration* und schaut euch die Klasse `Schleifendreher` an. Dort findet ihr Beispiele für die verschiedenen Schleifentypen in Java. Führt die Beispiele aus, um euch mit den Schleifen vertraut zu machen. **Setzt euch mit den Fragen auseinander, die in den Methoden-Kommentaren stehen!**

Für das Verständnis der Schleifen ist es hilfreich, die Beispiele im Debug-Modus von BlueJ auszuführen.

 7.1.2 Das folgende Diagramm zeigt, wie der Kontrollfluss durch eine `for`-Schleife wandert:



Zeichnet entsprechende Diagramme für die do-while-Schleife und die while-Schleife. **Schriftlich.**

- 7.1.3 Wenn es in Karel keine `repeat`-Schleife gäbe, dafür aber eine `for`-Schleife wie in Java, wie würde man dann den Befehl `moveAcrossWorld` implementieren, welcher 9 Schritte nach vorne geht? **Schriftlich.**

Aufgabe 7.2 Eigene Schleifen schreiben

- 7.2.1 Schaut euch die Klasse `TextAnalyse` des Projektes *Iteration* an. Dort gibt es eine vorgegebene Methode `istFrage(String text)`, die demonstriert, wie man die Länge eines Strings erhält und wie man auf einzelne Zeichen eines Strings zugreift. Probiert diese Methode interaktiv aus, indem ihr ein Exemplar von `TextAnalyse` erstellt und dann `istFrage` z.B. mit dem aktuellen Parameter "Wie geht's?" aufruft.

Vergleicht die Methoden `istFrage` und `istFrageKompakt`. Worin unterscheiden sie sich?

Was passiert, wenn ihr der Methode `istFrage` den leeren String als aktuellen Parameter übergebt, also `istFrage("")`? Implementiert eine Lösung, die diesen Fall sinnvoll behandelt.

- 7.2.2 Schreibt nun eine eigene Methode `int zaehleVokale(String text)`, die für einen gegebenen Text als Ergebnis liefern soll, wie viele Vokale er enthält. Für den String "hallo" soll die Methode beispielsweise eine 2 zurückgeben. Verwendet in der Implementierung einen Schleifenzähler, der bei 0 beginnt und alle Positionen des Strings durchläuft.

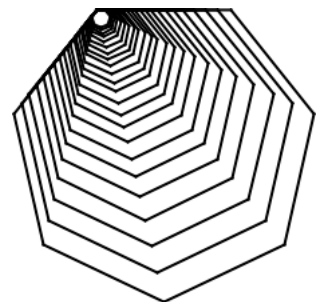
Die eigentliche Prüfung auf einen Vokal lässt sich am elegantesten mit der `switch`-Kontrollstruktur lösen. Schaut euch im Zweifel die Vorlesungsfolien an, wenn ihr nicht genau wisst, wie das `switch` funktioniert.

- 7.2.3 Schreibt eine weitere Methode `boolean istPalindrom(String text)`, die nur für Palindrome wie `anna`, `otto`, `regallager` oder `axa` `true` liefert. Vergleicht dazu die passenden Zeichen innerhalb des Strings.
- 7.2.4 Verwendet die Methode `toLowerCase()` aus der Klasse `String`, um den Unterschied zwischen Groß- und Kleinschreibung zu ignorieren, damit auch `Anna`, `Otto` und `Regallager` als Palindrome erkannt werden.

Aufgabe 7.3 Turtle Graphics

Kopiert das Projekt `TurtleGraphics` in euer Arbeitsverzeichnis und öffnet es in BlueJ. Das Projekt enthält zwei Klassen `Turtle` und `Dompteur`. Die Klasse `Turtle` stellt Methoden zur Verfügung, mit denen sehr einfach eine Turtle „bewegt“ werden kann; diese Bewegungen werden auf einer Zeichenfläche aufgezeichnet. Die Klasse `Dompteur` enthält eine Methode `start`, in der beispielhaft die Verwendung einer `Turtle` dargestellt ist. Mit Hilfe des Beispiels und der Dokumentation der `Turtle`-Schnittstelle sollen die folgenden Aufgaben gelöst werden.

- 7.3.1 Implementiert eine Methode in `Dompteur`, die mit Hilfe von `Turtle` ein n -Eck zeichnet. Die Anzahl der Ecken und die Kantenlänge sollen als Parameter übergeben werden.
- 7.3.2 Erweitert nun die Methode so, dass es möglich wird, die Position und Farbe des n -Ecks festzulegen.
- 7.3.3 Schreibt eine Methode in `Dompteur`, die kleiner werdende, ineinander geschachtelte n -Ecke zeichnet. Über Parameter soll festgelegt werden können,
- wie viele Ecken die n -Ecke haben sollen,
 - wie groß die Kantenlänge des ersten n -Ecks ist.



Aufgabe 7.4 Primzahlen

Primzahlen sind natürliche Zahlen, die genau zwei Teiler haben. Die ersten 6 Primzahlen lauten: 2, 3, 5, 7, 11, 13.

- 7.4.1 Legt ein neues BlueJ-Projekt namens *Primzahlen* an.
- 7.4.2 Schreibt darin eine neue Klasse namens `Primzahlen`.

- 7.4.3 Schreibt eine Methode `istTeilbar(int x, int y)`, welche prüft, ob x restlos durch y teilbar ist. Der Divisionsrest kann in Java mit dem Operator `%` bestimmt werden. Beispielsweise ist `123 % 10 == 3`.
- 7.4.4 Schreibt eine Methode `istPrimzahl(int x)`, welche auf Basis der soeben geschriebenen Methode `istTeilbar` überprüft, ob x eine Primzahl ist. Welche Teiler müssen hier überprüft werden?
- 7.4.5 Schreibt eine Methode `schreibePrimzahlenBis(int grenze)`, welche alle Primzahlen auf die Konsole schreibt, die kleiner als `grenze` sind.