TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK
Lehrstuhl für Angewandte Informatik / Kooperative Systeme
Prof. Dr. A. Brüggemann-Klein, R. Palenta, J. Kranz, Dr. A. Reuss

WS 15/16
Übungsblatt 6
20.11.2015

Praktikum Grundlagen der Programmierung

Abgabe: 30.11.2015 (vor 5:00 Uhr)

Hinweis: Implementieren Sie alle Hausaufgaben in einem Package namens **blatt06**.

Aufgabe 6.1 (P) Modellierung von einfachen Objekten

In dieser Aufgabe verwalten wir Namen als Objekte.

- Erstellen Sie eine Klasse `Name`, deren Objekte je einen String für den Vor- und den Nachnamen einer Person in privaten Attributen speichern. Beim Erzeugen eines Objekts der Klasse `Name` sollen die beiden Strings mit den entsprechenden Werten initialisiert werden.
- Statten Sie die Klasse mit den entsprechenden get- und set-Methoden für die beiden Attribute aus (`void setVorname(String newName)`, `String getNachname()` usw.).
- Erstellen Sie eine Methode `public String toString()`, die den Namen der Person zurückgibt. Für die Person Max Mustermann (Vorname Max, Nachname Mustermann) soll also z.B. "Max Mustermann" zurückgegeben werden.
- Verwenden Sie die `toString`-Methode, um eine Methode `print` zu schreiben, die beim Aufruf einen Text ausgibt, in dem der Name der Person enthalten ist, also beispielsweise "Hallo, mein Name ist Max Mustermann!".
- Fügen Sie eine Methode `printReversed` hinzu, welche ebenfalls einen Text ausgibt, der den Namen der Person enthalten soll, jedoch mit Nennung des Nachnamen zuerst, also z.B. "Hallo, mein Name ist Mustermann, Max.".
- Erstellen Sie zum Schluss eine Methode `boolean gleicheNamen(Name second)`, die zurückgibt, ob das übergebene `Name`-Objekt `second` den selben Vornamen und den selben Nachnamen hat. Diskutieren Sie mit ihrem Tutor/ihrer Tutorin, inwieweit mit dem Operator == `Name`-Objekte und `String`-Objekte verglichen werden können[1] und ob Sie diesen für die Lösung der Aufgabe verwenden können.
- Testen Sie die Klasse `Name` mit einem Hauptprogramm in der Client-Klasse `NameTest`, welches zwei `Name`-Objekte instanziiert und die Objekt-Methoden verwendet, um die Namen zu ändern und sie auszugeben.

Diskutieren Sie die folgenden Vorschläge zur Lösung von Teilen der Aufgabenstellung aus der Datei `BadName.java` von der Moodle-Praktikumsseite:

a) den Konstruktor

```
public void Name(String vorname, String nachname) {
    vorname = vorname;
    String nachname = nachname;
}
```

[1] `http://www.thejavageek.com/2013/07/27/string-comparison-with-equals-and-assignment-operator/`

b) die folgende Implementierung von `getVorname`:

```
public static String getVorname() {
    return vorname;
}
```

c) die folgende Signatur der Methode `print`:

```
private void print()
```

Aufgabe 6.2 (P) Listen, API

In der Vorlesung wurde die Java-API `LinkedList<E>` eingeführt, wobei E der Typ der Elemente ist, die in der Liste gespeichert werden. Beispiele sind also `LinkedList<String>` oder `LinkedList<Integer>`. Die LinkedList ist eine einfache Variante, mehrere Objekte eines Typs zusammenzufassen bzw. abzuspeichern. Dazu stehen viele nützliche Methoden zur Verfügung. Verschaffen Sie sich einen Überblick über diese Methoden: `http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html`. Schreiben Sie ein kleines Testprogramm, um die folgenden Methoden auf einer `LinkedList<Integer>` auszuprobieren:

- add(E element)
- add(int index, E element)
- addFirst(E element)
- addLast(E element)
- clear()
- clone()
- contains(Object o)
- get(int index)
- getFirst()
- getLast()
- indexOf(Object o)
- isEmpty()
- pop()
- push(E element)
- remove(int index)
- remove(Object o)
- set(int index, E element)
- size()

Implementieren Sie

- eine Methode `public static LinkedList<String> removeAll(LinkedList<String> list, String element)`, die alle Vorkommen des Strings `element` aus der Liste `list` löscht und die dadurch entstandene Liste zurückgibt.
- eine Methode `public static LinkedList<Integer> allOccurences(LinkedList<String> list, String element)`, die eine `LinkedList<Integer>` zurückgibt, die alle Indizes enthält, an deren Position der String `element` in der Liste `list` vorkommt. Falls `element` nicht in `list` vorkommt, wird eine leere Liste zurückgegeben.

Testen Sie Ihre implementierten Methoden.

Aufgabe 6.3 (P) OO - Was und wo

Diskutieren Sie, was im folgenden Programm passiert. Betrachten Sie dazu zunächst die Klassendefinitionen und überlegen sich dann, welche Statements in welcher Reihenfolge abgearbeitet werden.

```
1  import java.util.LinkedList;
2  class Howler {
3    private int    i;
4    private int    j;
5    private Walker walker;
6    public Howler(int i, int j, Walker b) {
7       setIandJandWalker(i, j, b);
8    }
9    public void setIandJandWalker(int i, int j, Walker newWalker) {
10      i      = i;
11      this.j = j;
12      walker = newWalker;
13   }
14   public String toString() {
15      return "Toll, ich bin eine Instanz von Howler und habe ein " +
16             "i=" + i + " und ein " +
17             "j=" + j + " und eine Referenz auf den Walker:\n\t" + walker;
18   }
19 }
20
21 class Walker {
22   private String name;
23   private int    i;
24   public Walker(String name, int i) {
25      this.i    = i;
26      this.name = name;
27   }
28   public String toString() {
29      return "Wow, ich bin eine Instanz von Walker und heisse " +
30             name + ". Ausserdem habe ich ein i=" + i;
31   }
32   public void increase() {
33      i++;
34   }
35   public boolean referencesSameName(String s) {
36      return (s == name);
37   }
38   public boolean equalsSameName(String s) {
39      return s.equals(name);
40   }
41 }
42
43 public class MyMainClass {
44   public static void main(String[] args) {
45      int i = 1, j = 2;
```

```
46        Walker w = new Walker("TestB", 0);
47        Howler h = new Howler(i, j, w);
48        System.out.println(h.toString());
49        System.out.println(h);
50        i++;
51        w.increase();
52        System.out.println(h.toString());
53        w = new Walker("hollaDieWaldfee", 0);
54        System.out.println(h.toString());
55        LinkedList<String> s = new LinkedList<>();
56        s.add("holla");
57        s.add("Die");
58        s.add("Waldfee");
59        System.out.println(w.referencesSameName(s.get(0) + s.get(1) + s.get
              (2)));
60        System.out.println(w.equalsSameName(s.get(0) + s.get(1) + s.get(2)));
61    }
62 }
```

Aufgabe 6.4 [7.5 Punkte] **(H) Terminalhaus III**

In dieser Aufgabe soll das bekannte Terminalhaus als Klasse implementiert werden. Implementieren Sie eine Klasse `Zeichenhaus.java`, die ein Terminalhaus repräsentiert. Das Terminalhaus soll durch die Gesamtbreite und die Breite des Innenraumes repräsentiert werden. Die Höhe des Mauerwerkes soll wie zuvor konstant sechs Zeichen hoch sein.

a) Erstellen Sie eine Klasse `Zeichenhaus` und deklarieren Sie die notwendigen Attribute zur Repräsentation des Zeichenhauses. Auf die Attribute der Klasse soll von außen nicht zugegriffen werden können.

b) Implementieren Sie einen Konstruktor, der als Parameter die Breite des Hauses und des Innenraumes üebergeben bekommt. Die Parameter sollen die gleichen Kriterien wie auf Aufgabenblatt 3 erfüllen. Sind die Kriterien nicht erfüllt, soll die Hausbreite auf 0 gesetzt werden. Der Konstruktor soll von außerhalb der Klasse zugreifbar sein.

c) Implementieren Sie eine Methode `dach()`, die das Dach des Hauses als `String` zurückgibt. Auf die Methode soll von außerhalb der Klasse nicht zugegriffen werden können.

d) Implementieren Sie eine Methode `mauer()`, die die Mauer des Hauses als `String` zuückgibt. Auf die Methode soll von außerhalb der Klasse nicht zugegriffen werden können.

e) Implementieren Sie eine Methode `toString()`, die das komplette Terminalhaus als `String` zurückgibt. Wurden beim Konstruktor ungültige Parameter eingegeben, soll statt einem Terminalhaus ein Hinweis auf diese ungültigen Parameter zurückgegeben werden. Auf die Methode soll von außerhalb der Klasse zugegriffen werden können.

f) Implementieren Sie eine Klasse `ZeichenhausClient`, in der Sie vier `Zeichenhaus`-Objekte mit verschiedenen Parametern initialisieren und anschließend auf der Konsole ausgeben. Die ersten drei Objekte sollen mit gültigen Parametern initialisiert werden, das letzte Objekt mit ungültigen.

Aufgabe 6.5 [10.5 Punkte] **(H) Schiffe versenken**

Es ist die folgende Variante des Spiels *Schiffe-Versenken* zu implementieren. Gespielt wird auf einem quadratischen Spielfeld der Seitenlänge 10, d.h. es gibt 100 einzelne Felder. Die Felder eines Spielfeldes werden über ihre Koordinaten nach rechts (0 bis 9) sowie nach oben (0 bis 9) identifiziert. Das Feld (0,0) liegt also links unten in der Ecke. Zu Beginn markiert der Computer auf dem Spielfeld *zufällig* Bereiche, die entweder aus horizontal benachbarten

Feldern bestehen oder aus vertikal benachbarten Feldern. Diese bezeichnen wir als *Schiffe*. Zwei Schiffe dürfen nicht aneinander angrenzen, auch nicht diagonal. D.h. zwischen zwei Feldern, welche zu unterschiedlichen Schiffen gehören, liegt immer mindestens ein Feld, welches zu keinem Schiff gehört. Es soll von den Längen 2, 3 und 4 jeweils ein Schiff platziert werden. Zum Platzieren der Schiffe können Sie die Methode `myRandom` aus der vorgegebenen Klasse `Field` verwenden, die Zufallszahlen zwischen `low` (inklusive) und `high` (exklusive) erzeugt.

Im Verlauf des Spiels rät der Spieler wiederholt *ein Feld*, auf dem er ein Schiff vermutet. Sind alle Felder eines Schiffs geraten („getroffen“), gilt das Schiff als *gesunken*. Das Spiel endet, wenn alle Schiffe gesunken sind. Das Programm gibt nach jedem Raten das Spielfeld aus und markiert darauf, welche Felder einen Treffer ergaben, welche Felder bereits Teil eines gesunkenen Schiffs sind und welche Felder bisher ohne Treffer (Schuss ins Wasser) geraten wurden.

Zum Testen des Programms soll es während des Spielverlaufs alternativ zum Raten eine Möglichkeit geben, sich die Platzierung der Schiffe anzeigen zu lassen. Z.B. kann diese Ausgabe bei Eingabe von ungültigen Koordinaten (außerhalb vom Spielfeld) anstelle eines Fehlerhinweises erfolgen.

Implementieren Sie das Spiel objektorientiert! Verwalten Sie das Spielfeld, die Spielfeldkoordinaten und die Schiffe jeweils in einer eigenen Klasse.

Hinweis: Nennen Sie Ihr Programm `Battleship.java`. Die Praktikums-Webseite bietet eine Vorlage, die Sie nutzen können und bereits eine Spielfeldausgabe enthält. Denken Sie daran, Ihr Programm ausreichend zu kommentieren, insbesondere wenn Sie die Vorlage nicht nutzen.

Tipps: Verwenden Sie zum zufälligen Platzieren eines Schiffs eine Schleife, welche immer wieder zufällig eine Position erzeugt und verlassen wird, sobald die Schiffsposition erlaubt ist.

Erweiterung: Studierende mit fortgeschrittenen Kenntnissen können ihr Spiel auf zwei Spieler erweitern, die abwechselnd Schiffe des anderen Spielers raten. Es muss jedoch die Möglichkeit geben, nur die in der Angabe geforderte Variante zu spielen, und es wird auch nur diese bewertet.