Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE24-KlassenUndObjekteFort
(Stand 28.09.2012)

Aufgabe 1:

Implementieren Sie eine Klasse *Student*. Die Klasse soll eine Methode *immatrikulieren* haben, der das Alter eines Studenten als Parameter übergeben wird, sowie eine Methode *exmatrikulieren*, sowie eine dritte Methode, die bei ihrem Aufruf den Altersdurchschnitt aller gerade immatrikulierten Studenten liefert.

Aufgabe 2:

Schauen Sie sich folgendes Programm an. Die Klasse `IO` ist die aus der Vorlesung bekannte Klasse:

```
class Addieren {

  public static void main(String[] args) {
      IO.println(new Int(8).add(IO.readInt()).add(-2).makeString());
  }
}
```

Aufgabe: Entwickeln Sie eine Klasse `Int`, so dass sich das Programm kompilieren lässt. Die Klasse `Int` soll dabei folgende Semantik haben:

- Dem Konstruktor wird ein int-Wert übergeben, der intern gespeichert werden soll.
- Bei Aufruf der Methode `add` für ein Int-Objekt soll der übergebene int-Wert zum internen Wert addiert und anschließend eine Referenz auf das Objekt selbst geliefert werden.
- Die Methode `makeString` soll so implementiert werden, dass der interne int-Wert in seiner String-Repräsentation geliefert wird.

Bei einer Benutzereingabe von 3 soll das Programm bspw. „9“ auf den Bildschirm ausgeben, bei einer Eingabe von -7 „-1“.

Aufgabe 3:

Definieren Sie eine Klasse zur Repräsentation eines UKW-Radios. Der Zustand des Radios ist gegeben durch eine Frequenz zwischen 87.5 und 108.0 MHz. Nach dem Erzeugen des Radio-Objektes beträgt die Frequenz 87.5 MHz. Die Frequenz kann in den angegebenen Frequenzen schrittweise um 0.5 MHz nach oben bzw. unten verändert werden. Weiterhin stehen *n* Stationstasten zur Verfügung, auf denen man Frequenzen speichern bzw. gespeicherte Frequenzen wieder abrufen kann.

Insgesamt soll Ihre Klasse also die folgenden Methoden definieren:

- Einen Konstruktor, dem die Anzahl *n* an Stationstasten als Parameter übergeben wird
- Eine Methode, die die aktuelle Frequenz liefert
- Eine Methode zum Verringern der Frequenz um 0.5 MHZ
- Eine Methode zum Erhöhen der Frequenz um 0.5 MHZ
- Eine Methode, um die aktuelle Frequenz auf einer Stationstaste zu speichern
- Eine Methode, um die aktuelle Frequenz auf die Frequenz einer angegebenen Stationstaste einzustellen

Im StudIP steht ein Testprogramm mit einer GUI für Ihre Klasse zur Verfügung. Wenn Ihre Klasse korrekt implementiert ist und sich in das Testprogramm ohne Compilierfehler einbinden lässt, sollte beim Start des Testprogramms folgende interaktive GUI auf dem Bildschirm erscheinen.

Aufgabe 4:

Gegeben seien folgende Java-Klassen:

```
class Person {

      // liefert den Namen der Person
      public String getName()
}

class Buch {

      // liefert die ISBN-Nummer des Buches
      public String getISBN()

      // liefert den Titel des Buches
      public String getTitel()

      // liefert die Autoren des Buches
      public Person[] getAutoren()

      // liefert den Verkaufspreis des Buches
      public double getPreis()
}
```

```
class Buchhandlung {

      // privater Konstruktor; Umsetzung des Singleton-Musters
      private Buchhandlung()

      // Klassenmethode, die zum Namen einer Buchhandlung das
      // entsprechende Buchhandlung-Objekt liefert;
      // bei ungültigem Namen wird null geliefert
      public static Buchhandlung getBuchhandlung(String name)

      // liefert die Anzahl an verkauften Exemplaren in dieser
      // Buchhandlung für das angegebene Buch
      public int getAnzahlVerkaufteExemplare(Buch buch

      // liefert die Bücher, die die Buchhandlung im Sortiment hat
      public Buch[] getBuecherImSortiment()
}
```

Schreiben Sie auf der Basis dieser Klassen ein Programm, das folgendes tut:

Zunächst wird der Nutzer nach dem Namen einer Buchhandlung und dem Namen eines Autors gefragt. Anschließend berechnet das Programm, wie viele Tantiemen der angegebene Autor für seine Bücher bekommt, die in der angegebenen Buchhandlung verkauft worden sind, und gibt die Summe auf den Bildschirm aus. Als Tantiemen bekommen die Autoren eines Buches dabei 10 Prozent des Buchpreises jedes verkauften Exemplares, die Sie evtl. (gleichmäßig) miteinander teilen müssen.

Aufgabe 5:

Gegeben seien folgende Java-Klassen:

```
public class Person {
      // liefert den Namen der Person
      public String getName()
}

public class Student extends Person {

      // nicht von außen direkt instantiierbar
      private Student()

      // liefert den Studenten mit der angegebenen Matrikelnummer
      public static Student getStudent(String matrikelnummer)
}

public class PruefungsErgebnis {
      // liefert den Namen des Moduls der Prüfung
      public String getModulname()

      // liefert alle Professoren, die für die Prüfung zuständig
waren
      public Person[] getProfessoren()
```

```
	// liefert den Studenten mit diesem Prüfungsergebnis
	public Student getStudent()

	// liefert die Note der Prüfung
	public int getNote()
}

public class PruefungsDB {
	// nicht von außen direkt instantiierbar
	private PruefungsDB()

	// liefert die Prüfungsdatenbank
	public static PruefungsDB getPruefungsDB()

	// liefert alle Prüfungsergebnisse des angegebenen Studenten
	public PruefungsErgebnis[] getNoten(Student student)
}
```

Schreiben Sie auf der Basis dieser Klassen ein Programm, das folgendes tut:

Zunächst wird der Nutzer nach dem Namen eines Professors und der Matrikelnummer eines Studierenden gefragt. Anschließend berechnet das Programm den Notendurchschnitt des entsprechenden Studierenden bei Prüfungen mit dem entsprechenden Professor und gibt den Notendurchschnitt auf den Bildschirm aus.

Aufgabe 6:

Ein ASCII-Bildschirm im Sinne dieser Aufgabe ist ein rechteckiges Gebilde, das aus einer Menge an Kacheln besteht. In diese Kacheln können char-Zeichen ausgegeben werden.

Die folgende Klasse `Kachel` repräsentiert entsprechende Bildschirm-Kacheln.

```
// Repaesentiert eine Zelle eines ASCII-Bildschirms
public class Kachel {

	int reihe;
	int spalte;

	// initialisiert die angegebene Kachel mit einem Leerzeichen
	public Kachel(int reihe, int spalte) {
		this.reihe = reihe;
		this.spalte = spalte;
	 // Implementierung hier unwichtig
	}

	// Zeichnet das uebergebene Zeichen in die entsprechende Kachel
	public void zeichne(char zeichen) {
		// Implementierung hier unwichtig
	}
```

```
}
```

Die folgende Klasse `Bildschirm` repräsentiert entsprechende Bildschirme.

```
// Repraesentiert einen rechteckigen ASCII-Bildschirm, der aus
einzelnen
// Kacheln besteht
public class Bildschirm {

      // erzeugt/initialisiert einen Bildschirm der angegebenen
Groesse
      public Bildschirm(int anzahlReihen, int anzahlSpalten) {    }

      // liefert die Kacheln der angegebenen Reihe
      // Hinweis: Sie muessen sich nicht um Index-Fehler kuemmern
      public Kachel[] getReihe(int reihenIndex) {      }

      // liefert die Kacheln der angegebenen Spalte
      // Hinweis: Sie muessen sich nicht um Index-Fehler kuemmern
      public Kachel[] getSpalte(int spaltenIndex) {    }
```

Teilaufgabe (a)
Implementieren Sie die Klasse Bildschirm, d.h. definieren Sie geeignete interne Datenstrukturen sowie implementieren Sie die aufgeführten Konstruktoren und Methoden. Hinweis: Es dürfen keine zusätzlichen public-Methoden definiert werden.

Teilaufgabe (b)
Entwickeln Sie auf der Grundlage der Klassen Kachel und Bildschirm ein Programm, das ein Rechteck der Größe *n* (*n* ist eine int-Konstante mit Werten größer 0) auf einen Bildschirm mit 24 Zeilen und 80 Spalten zeichnet (beginnend in der linken oberen Ecke des Bildschirms). Dabei dürfen nur die public-Konstruktoren und -Methoden der Klassen genutzt werden. Der Zugriff auf in den Klassen definierte Attribute ist ebenfalls nicht erlaubt.

Beispiel (n = 2):

```
 --
|  |
|  |
 --
```

Beispiel (n = 5):

```
 -----
|     |
|     |
|     |
|     |
|     |
 -----
```

Aufgabe 7:

Gegeben seien folgende Klassen:

```
class FachNote {
	private String fach; // das Fach (Deutsch, Mathe, ...)
	private int note; // die Note in dem Fach (1 .. 6)

	public FachNote(String fach, int note) {
		this.fach = fach;
		this.note = note;
	}

	public String getFach() {
		return this.fach;
	}

	public int getNote() {
		return this.note;
	}
}

class Schueler {
	private String name; // Name des Schuelers
	private FachNote[] noten; // Noten in den einzelnen Faechern

	public Schueler(String name, FachNote[] noten) {
		this.name = name;
		this.noten = noten;
	}

	public String getName() {
		return this.name;
	}

	public FachNote[] getNoten() {
		return this.noten;
	}
}

class NichtBelegtesFachException extends Exception {
	private String fach; // Name des nicht belegten Faches

	public NichtBelegtesFachException(String fach) {
		this.fach = fach;
	}

	public String getFach() {
		return this.fach;
	}
}
```

Implementieren Sie auf der Grundlage dieser Klassen die folgende Funktion:

```
	/**
```

```
     * Berechnet die Durchschnittsnote der uebergebenen Schueler in dem
     * uebergebenen Fach.
     * Hinweis: nicht alle Schueler muessen dieselben Faecher
     * belegen.
     *
     * @param schueler Menge an Schuelern (!= null)
     * @param fach Name des Faches (!= null)
     * @return die Durchschnittsnote der uebergebenen Schueler in dem
     *         uebergebenen Fach
     * @throws NichtBelegtesFachException
     *      wird geworfen, wenn kein Schueler das uebergebene Fach belegt
     */
    static double durchschnittsnote(Schueler[] schueler, String fach)
                throws NichtBelegtesFachException
```

Aufgabe 8:

Beim Online-Banking muss sich der Bankkunde zunächst mit seiner Kontonummer und einer persönlichen Identifikationsnummer (PIN) am Bankrechner authentifizieren. Für das Auslösen sicherungspflichtiger Aufträge (z. B. einer Überweisung) muss außerdem eine Transaktionsnummer (TAN) übermittelt werden. Diese wird aus einer Liste von 100 TANs entnommen, die dem Kunden vorher zugeschickt und durch ihn freigegeben wurde.

An die Transaktionsnummern werden folgende Bedingungen gestellt:

- Eine TAN ist eine zufällige 6-ziffrige Zahl zwischen 100000 und 999999.
- Jede TAN kommt nur einmal vor.
- Jede TAN darf nur einmal verwendet werden.

Folgendes ist eine gültige TAN-Liste:

```
963002 701837 543423 239150 526763 924103 396054 411076 977108 857169
255549 921595 447899 626258 781718 527762 802132 157715 797227 781895
821065 172984 527924 801503 820360 993706 631421 520026 454031 869184
914790 647546 166143 805788 514630 531489 548688 699481 397882 154254
138872 724491 488188 712325 415943 665750 728851 511410 248348 253840
626526 370290 910463 951473 554520 500185 608195 761195 107366 297942
100456 958189 774141 589685 361569 620076 901807 299068 598361 838014
378822 368426 238388 835745 121001 578457 852160 497733 155579 560223
219681 835335 844747 808295 524834 683175 212941 301816 292159 620207
297252 120304 248315 410165 332899 947068 713189 278126 422161 983132
```

Aufgabe: Implementieren Sie eine Klasse `TANListe` zur Verwaltung entsprechender Transaktionsnummern.

Überlegen Sie sich zunächst eine Datenstruktur zur Speicherung von 100 TANs. Dabei sollen sowohl der Wert jeder TAN als auch ihr Zustand (verbraucht/nicht verbraucht) erfasst werden.

Implementieren Sie dann die folgenden Methoden:

1. Einen *Konstruktor*, in dem die 100 nicht-verbrauchten TANs der TAN-Liste erzeugt und gespeichert werden.
2. Eine boolesche Methode `verbrauchen` mit einem int-Wert als Parameter. Die Methode soll prüfen, ob (im Parameter) eine gültige, nicht verbrauchte TAN der TAN-Liste übermittelt wurde. In diesem Fall soll die TAN in der Liste als

verbraucht markiert sowie der Rückgabewert `true` geliefert werden. In allen anderen Fällen soll `false` zurückgegeben werden.

3. Eine Methode `print`, durch die die noch nicht verbrauchten TANs der TAN-Liste auf den Bildschirm ausgegeben werden.

Aufgabe 9:

Langtons Ameise ist eine Turingmaschine mit einem zweidimensionalen Speicher und wurde 1986 von Christopher Langton entwickelt. Sie ist ein Beispiel dafür, dass ein deterministisches (das heißt nicht zufallsbedingtes) System aus einfachen Regeln sowohl für den Menschen visuell überraschend ungeordnet erscheinende als auch regelmäßig erscheinende Zustände annehmen kann. (Quelle: Wikipedia)

Ausgangspunkt:

Gegeben sei eine quadratische 2-dimensionale Welt mit SIZE * SIZE Zellen. SIZE (hier SIZE = 200) sei die Anzahl der Reihen und Spalten. Die Zellen können zwei Zustände einnehmen (weiß und schwarz). Anfangs sind alle Zellen weiß. In dieser Welt lebt genau eine Ameise. Anfangs befindet sich die Ameise auf der Zelle in der Mitte der Welt und schaut nach Norden. Nun wird STEPS (hier STEPS = 40000) mal folgender „Ameisenalgorithmus“ ausgeführt:

- Befindet sich die Ameise auf einer weißen Zelle, so färbt sie sie schwarz, dreht sich um 90 Grad nach rechts und begibt sich auf die nächste Zelle in der neuen Blickrichtung.
- Befindet sich die Ameise auf einer schwarzen Zelle, so färbt sie sie weiß, dreht sich um 90 Grad nach links und begibt sich auf die nächste Zelle in der neuen Blickrichtung.

Die Welt sei dabei ein Torus, d.h. wenn die Ameise die Welt nach oben verlässt, erscheint sie wieder unten und umgekehrt; wenn sie die Welt nach links verlässt, erscheint sie wieder rechts und umgekehrt.

Vorgabe:

Gegeben Sei folgendes Programm:

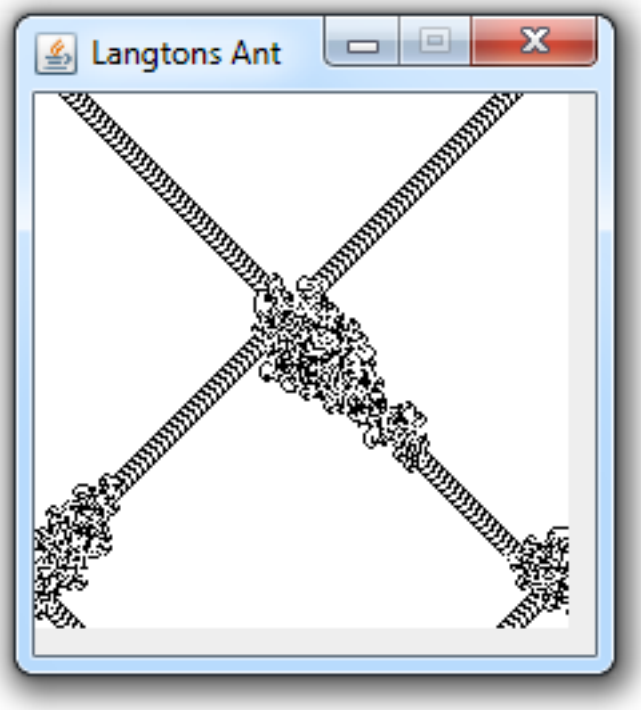
```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.image.BufferedImage;

import javax.swing.JFrame;
import javax.swing.JPanel;

public class LangtonAnt {

	private static final int SIZE = 200;
	private static final int STEPS = 40000;

	public static void main(String[] args) {
		javax.swing.SwingUtilities.invokeLater(new Runnable() {
			@Override
			public void run() {
				// erzeugt eine Welt der Groesse SIZE * SIZE
```

```
                        World world = new World(LangtonAnt.SIZE);

                        AntWorldPanel worldPanel = new AntWorldPanel(world);
                        AntFrame frame = new AntFrame(worldPanel);
                        frame.setLocation(100, 100);
                        frame.setResizable(false);
                        frame.setVisible(true);
                        AntSimulation simulation = new AntSimulation(world,
                                        LangtonAnt.STEPS, worldPanel);
                        simulation.start();
                    }
                });
        }
}

class AntSimulation extends Thread {

        private World world;
        private int steps;
        private AntWorldPanel panel;

        AntSimulation(World world, int steps, AntWorldPanel panel) {
                this.world = world;
                this.steps = steps;
                this.panel = panel;
        }

        @Override
        public void run() {
                try {
                        for (int i = 0; i < this.steps; i++) {

                                // holt sich die Ameise aus der Welt
                                Ant ant = this.world.getAnt();

                                // fuehrt einmal den Ameisenalgorithmus aus
                                ant.nextStep();

                                if (i % 30 == 0) {
                                        javax.swing.SwingUtilities.invokeAndWait(new
Runnable() {
                                                @Override
                                                public void run() {
                                                        AntSimulation.this.panel.repaint();
                                                }
                                        });
                                }
                        }
                } catch (Exception exc) {
                        exc.printStackTrace();
                }

        }
}

class AntFrame extends JFrame {

        AntFrame(AntWorldPanel world) {
                super("Langtons Ant");
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setLayout(new BorderLayout());
                this.add(world);
                this.pack();
        }
}

class AntWorldPanel extends JPanel {

        private World world;
        private BufferedImage image;
```

```
	AntWorldPanel(World world) {
		this.world = world;
		this.setPreferredSize(new Dimension(world.getSize(),
world.getSize()));
		this.image = new BufferedImage(world.getSize(), world.getSize(),
				BufferedImage.TYPE_INT_ARGB);
	}

	@Override
	protected void paintComponent(Graphics g) {
		super.paintComponent(g);

		Graphics buffer = this.image.getGraphics();
		buffer.setColor(Color.WHITE);
		buffer.fillRect(0, 0, this.image.getWidth(), this.image.getHeight());

		for (int row = 0; row < this.world.getSize(); row++) {
			for (int column = 0; column < this.world.getSize(); column++) {

				// liefert true, wenn die Zelle in Zeile row und
				// Spalte column sich im Zustand schwarz befindet
				if (this.world.isCellBlack(row, column)) {
					buffer.setColor(Color.BLACK);
				} else {
					buffer.setColor(Color.WHITE);
				}
				buffer.fillRect(column, row, 1, 1);
			}
		}
		g.drawImage(this.image, 0, 0, null);

	}
}
```

Aufgabe:

Implementieren Sie die fehlenden Klassen `World` und `Ant`, die die Welt bzw. die Ameise repräsentieren. Wenn Sie alles korrekt gemacht haben, sollte innerhalb einiger Sekunden nach und nach das folgende Fenster auf dem Bildschirm erscheinen.



Hinweise:

- Sie brauchen den vorgegebenen Code nicht komplett zu verstehen; schauen Sie sich nur die Stellen an, an denen die Klassen `World` bzw. `Ant` genutzt werden. Hier stehen entsprechende Kommentare. Insbesondere brauchen Sie aus den Klassen `World` und `Ant` nicht auf die gegebenen Klassen zugreifen!

- Sie dürfen die vorgegebenen Klassen nicht ändern!