Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE23-DefinitionVonKlassen
(Stand 16.12.2011)

Aufgabe 1:

Laden Sie die Klasse „Shapes.java“ in Eclipse.

- Ändern Sie die Konstruktoren so ab, dass als Parameter die anfängliche Farbe und Position der Objekte übergeben wird.
- Fügen Sie zwei Methoden setVisible und setInvisible ein. Nur Visible-gesetzte Objekte sollen auf dem Bildschirm sichtbar sein.
- Erweitern Sie das Beispiel um eine Klasse Rectangle. Kopieren Sie dazu die Klasse Square und nehmen Sie notwendige Änderungen vor.
- Schreiben Sie ein kleines Testprogramm, das eine rudimentäre Ampel auf den Bildschirm zeichnet.

Aufgabe 2:

Der *Body-Mass-Index* (*BMI*) ist eine Maßzahl für die Bewertung des Körpergewichts eines Menschen. Der BMI berechnet sich aus dem Körpergewicht [kg] dividiert durch das Quadrat der Körpergröße [m^2]. Die Formel lautet: BMI = Körpergewicht / (Körpergröße in m)^2. Die Einheit des BMI ist demnach kg/m^2.

Der "wünschenswerte" BMI hängt vom Alter ab. Folgende Tabelle zeigt BMI-Werte für verschiedene Altersgruppen:

Alter	BMI
<= 24 Jahre	19-24
25-34 Jahre	20-25
35-44 Jahre	21-26
45-54 Jahre	22-27
55-64 Jahre	23-28
>= 65 Jahre	24-29

Liegt der BMI unterhalb der angegebenen Werte hat die Person Untergewicht, darüber Übergewicht.

Definieren Sie analog zu dem Beispiel in der Unterrichtseinheit 23 eine Klasse `Mensch`, für deren Objekte abgefragt werden kann, ob der entsprechende Mensch Unter-, Normal- oder Übergewicht hat. Grundlage ist dabei diesmal der BMI. Testen Sie Ihre Klasse an dem folgenden Beispielprogramm:

```
public class Aufgabe2 {
	public static void main(String[] args) {
		Mensch person = new Mensch(IO.readInt("Alter: "),
				IO.readDouble("Gewicht (kg) eingeben: "),
				IO.readDouble("Groesse (m) eingeben: "));

		if (person.hatUebergewicht()) {
			IO.println("Na, du alter Fettsack!");
		} else if (person.hatUntergewicht()) {
			IO.println("Na, du Hungerhaken!");
		} else {
			IO.println("Idealer Gewichtsbereich!");
		}
		IO.println("Dein BMI ist " + person.getBMI());
	}

}
```

Aufgabe 3:

Schauen Sie sich das folgende Programm an:

```
class Rabatte {

	public static void main(String[] args) {
		double preis = 100.0; // Euro
		double kleinRabatt = 10.0; // Prozent
		double grossRabatt = 15.0; // Prozent

		// Erzeugen eines Haendlers mit Preis- und
		// Rabattinformationen
		Haendler haendler =
		  new Haendler(preis, kleinRabatt, grossRabatt);

		int anzahlKaeufe = IO.readInt("Anzahl Kaeufe: ");
		for (int i = 0; i < anzahlKaeufe; i++) {
			int menge = IO.readInt("Zu kaufende Menge: ");
			// eine bestimmte Menge beim Haendler einkaufen;
			// geliefert werden die jeweiligen Kosten
			double kosten = haendler.kaufen(menge);
			IO.println("Anfallende Kosten: " + kosten);
		}

		// Ausgabe der vom Haendler gesamt erzielten Einnahmen
		IO.println("Einnahmen: " +
				haendler.liefereGesamteEinnahmen());

		// Ausgabe der vom Haendler gesamt gewaehrten Rabatte;
		// (also "Rabattverluste")
		IO.println("Rabatte: " +
				haendler.liefereGesamtGegebeneRabatte());
	}

}
```

In diesem Programm wird zunächst ein Händler erzeugt, der ein einzelnes Produkt zu einem angegebenen Preis verkauft und dabei seinen Kunden Mengenrabatte einräumt:

- Bei einem Kaufvorgang bis zu einer Menge von 5 Produkten (einschließlich) wird kein Rabatt gewährt.
- Bei einem Kaufvorgang zwischen 6 und 10 Produkten (einschließlich) wird ein "kleinRabatt" (in Prozent) gewährt.
- Bei einem Kaufvorgang über 10 Produkten wird ein "grossRabatt" (in Prozent) gewährt.

Anschließend werden unter diesen Bedingungen eine bestimmte Anzahl an Kaufvorgängen durchgeführt und die jeweils anfallenden Kosten ausgegeben. Zum Schluss werden die gesamten Einnahmen sowie die insgesamt gewährten Rabatte des Händlers ausgegeben.

Aufgabe: Implementieren Sie eine Klasse *Haendler*, so dass sich das obige Programm compilieren lässt und die beschriebene Semantik erfüllt wird. Sie dürfen das gegebene Programm natürlich nicht verändern.

Im Folgenden wird ein Beispiel für eine mögliche Ausgabe des Programms gegeben (in <> die Eingaben des Benutzers):

```
Anzahl Kaeufe: <3>
Zu kaufende Menge: <5>
Anfallende Kosten: 500.0
Zu kaufende Menge: <7>
Anfallende Kosten: 630.0
Zu kaufende Menge: <12>
Anfallende Kosten: 1020.0
Einnahmen: 2150.0
Rabatte: 250.0
```

Aufgabe 4:

Ein Wörterbuch im Sinne der folgenden Aufgabe ist dadurch charakterisiert, dass es mehrere Begriffe enthält, denen jeweils eine einzelne Übersetzung zugeordnet ist.

Teilaufgabe (a): Implementieren Sie eine Klasse `Woerterbuch` mit folgenden Methoden:

1. Einen Konstruktor, der die maximale Anzahl an zu speichernden Begriffen als Parameter übergeben bekommt.
2. Eine Methode `einfuegen` mit zwei `String`-Parametern, nämlich einem Begriff und einer Übersetzung. Existiert der übergebene Begriff bereits im Wörterbuch, soll die alte Übersetzung durch die neue überschrieben werden.
3. Eine Methode `getUebersetzung`, die einen String als Parameter übergeben bekommt und die im Wörterbuch gespeicherte Übersetzung dieses Begriffs als Funktionswert (`String`) liefert.

Teilaufgabe (b): Schreiben Sie mit Hilfe der Klasse `Woerterbuch` ein kleines Programm, in dem ein Benutzer zunächst eine Menge an Begriffen und Übersetzungen eingeben kann und er sich anschließend zu einzelnen einzugebenden Begriffen die gespeicherten Übersetzungen ausgeben lassen kann.

Aufgabe 5:

Definieren Sie eine Klasse *Wuerfel* mit einer Methode *wuerfeln*. Die Methode soll Zufallszahlen zwischen 1 und 6 erzeugen und liefern. Nutzen Sie dazu die Funktion Math.random(), die zufällig double-Werte zwischen 0 und 1 (ausschließlich) generiert (`double zufall = Math.random();`)

Schreiben Sie weiterhin ein Programm, in dem zunächst ein Würfel erzeugt wird. Dieser wird anschließend 100.000 Mal gewürfelt. Geben Sie danach auf den Bildschirm aus, wie oft die einzelnen Zahlen gewürfelt worden sind.

Aufgabe 6:

Definieren Sie eine Klasse *LottoMaschine* mit einer Methode *naechsteZahl*. Die Methode soll Zufallszahlen zwischen 1 und 49 erzeugen und liefern. Aber Achtung: Wie bei einer richtigen Lottoziehung darf jede Zahl nur einmal vorkommen. Nutzen Sie dazu die Funktion Math.random(), die zufällig double-Werte zwischen 0 und 1 (ausschließlich) generiert (`double zufall = Math.random();`)

Schreiben Sie weiterhin ein Programm, das die Ziehung der Lottozahlen (6 Zahlen plus eine Zusatzzahl) simuliert und die Zahlen auf den Bildschirm ausgibt.

Aufgabe 7:

In dieser Aufgabe geht es um die Verschlüsselung von Texten. Klartexte über einem Klartextalphabet werden durch die Anwendung von Verschlüsselungsalgorithmen in Geheimtexte über einem Geheimtextalphabet überführt. Verschlüsselungsverfahren sollen gewährleisten, dass nur Befugte bestimmte Botschaften lesen können. Das Chiffrieren ist ein Verschlüsselungsverfahren, bei dem jeder Buchstabe in einem Text durch einen anderen Buchstaben ersetzt wird.

In dieser Aufgabe sollen Sie eine Klasse *CaesarVerschluesselung* implementieren, die zwei Methoden definiert:

```
class CaesarVerschluesselung{
     String chiffrieren(String botschaft);
     String dechiffrieren(String botschaft);
}
```

Die Methode `chiffrieren` soll eine entsprechende Umsetzung der übergebenen Zeichen vornehmen. Die Methode `dechiffrieren` bildet die reverse Funktion.

Die Caesar-Verschlüsselung beruht dabei auf einem Geheimtextalphabet, das um eine bestimmte *Stellenzahl n* gegenüber dem Klartextalphabet verschoben ist. Beispiel für n = 3: a -> d, b -> e, c -> f, ..., w -> z, x -> a, y -> b, z -> c. Chiffriert werden sollen hier nur Kleinbuchstaben. Alle anderen Zeichen sollen unverändert zurückgegeben werden. Implementieren Sie neben den beiden Methoden einen Konstruktor, dem die *Stellenzahl* als Parameter übergeben wird (Schlüssel).

Hinweis: Die Klasse java.lang.String stellt eine Methode `char charAt(int index)` zur Verfügung, die den Charakter an der index-ten Stelle liefert.

Implementieren Sie weiterhin ein Testprogramm (= Aufruf aller Methoden) für die Klasse *CaesarVerschluesselung*.

Aufgabe 8:

In dieser Aufgabe geht es wieder um die Verschlüsselung von Texten. Klartexte über einem Klartextalphabet werden durch die Anwendung von Verschlüsselungsalgorithmen in Geheimtexte über einem Geheimtextalphabet überführt. Verschlüsselungsverfahren sollen gewährleisten, dass nur Befugte bestimmte Botschaften lesen können.

Die Vignere-Verschlüsselung basiert auf dem so genannten Vignere-Quadrat:

```
Klar   a b c d e f g h i j k l m n o p q r s t u v w x y z

0      a b c d e f g h i j k l m n o p q r s t u v w x y z
1      b c d e f g h i j k l m n o p q r s t u v w x y z a
2      c d e f g h i j k l m n o p q r s t u v w x y z a b
3      d e f g h i j k l m n o p q r s t u v w x y z a b c
…
11     l m n o p q r s t u v w x y z a b c d e f g h i j k
…
24     y z a b c d e f g h i j k l m n o p q r s t u v w x
25     z a b c d e f g h i j k l m n o p q r s t u v w x y
```

Zeile 1 des Quadrates enthält ein Geheimtextalphabet mit einer Caesar-Verschiebung von 1; Zeile 2 des Quadrates enthält ein Geheimtextalphabet mit einer Caesar-Verschiebung von 2; usw.

Vorteil der Vignere-Verschlüsselung ist, dass eine Häufigkeitsanalyse zum Knacken des Geheimcodes versagt. Bei der Vignere-Verschlüsselung geht man nämlich so vor, dass man jeden Buchstaben einer geheim zu haltenden Botschaft anhand einer anderen Zeile des Vignere-Quadrates verschlüsselt. Um die Botschaft wieder entschlüsseln zu können, muss der Empfänger allerdings wissen, welche Zeile des Vignere-Quadrates für den jeweiligen Buchstaben benutzt wurde. Deshalb tauschen Sender und Empfänger vorher ein Schlüsselwort aus.

Nehmen wir einmal an, die zu verschlüsselnde Botschaft sei „truppenabzugnachosten“ und das Schlüsselwort sei „licht“. Zunächst wird das

Schlüsselwort über die Botschaft geschrieben und so lange wiederholt, bis jeder Buchstabe der Botschaft mit einem Buchstaben des Schlüsselwortes verknüpft ist.

```
Schlüsselwort     lichtlichtlichtlichtl
Klartext          truppenabzugnachosten
Geheimtext        ezwwipvcisfoehvswuaxy
```

Der Geheimtext wird dann folgendermaßen erzeugt: Um den ersten Buchstaben „t“ zu verschlüsseln, stellen wir zunächst fest, dass über ihm der Buchstabe „l“ steht, der wiederum auf eine bestimmte Zeile des Vignere-Quadrates verweist. Die mit „l“ beginnenden Reihe 11 enthält das Geheimtextalphabet, das wir benutzen, um den Stellvertreter des Klarbuchstabens „t“ zu finden. Also folgen wir der Spalte unter „t“ bis zum Schnittpunkt mit der Zeile „l“, und dort befindet sich der Buchstabe „e“ im Geheimtext. Genauso gehen wir mit allen weiteren Buchstaben der Botschaft vor.

Konkrete Aufgabe: Implementieren Sie eine Klasse VignereVerschlüsselung mit den folgenden Methoden:

```
class VignereVerschluesselung{
     String chiffrieren(String botschaft);
     String dechiffrieren(String botschaft);
}
```

Verschlüsselt werden sollen nur Kleinbuchstaben. Alle anderen Buchstaben sollen unverändert bleiben. Definieren Sie neben den beiden Methoden einen Konstruktor, dem das Schlüsselwort übergeben wird und der weiterhin das Vignere-Quadrat erzeugt. Dabei können Sie davon ausgehen, dass das übergebene Schlüsselwort nur Kleinbuchstaben enthält.

Implementieren Sie weiterhin ein Testprogramm (= Aufruf aller Methoden) für die Klasse *VignereVerschluesselung*.

Aufgabe 9:

Sicher haben Sie in Ihrer Kindheit mal das Spiel *Superhirn* (*Mastermind*) gespielt. Es ist ein Spiel für zwei Personen. Der eine Spieler wählt zu Beginn einen vierstelligen Farbcode aus, wobei sechs verschiedene Farben zur Verfügung stehen, die auch mehrfach verwendet werden können. Der andere Spieler versucht den Code herauszufinden. Als Hilfestellung bekommt er nach jedem Zug die Information, wie viele Stifte er in Farbe und Position richtig gesetzt hat, und wie viele Stifte zwar die richtige Farbe haben, aber an einer falschen Position stehen. Ein Treffer in Farbe und Position wird durch einen schwarzen Stift angezeigt, ein lediglich in der Farbe übereinstimmender Stift wird durch einen weißen Stift angezeigt (siehe auch http://de.wikipedia.org/wiki/Mastermind)

Die 6 Farben ersetzen wir in dieser Aufgabe durch die Ziffern 1 bis 6.

Implementieren Sie eine Klasse *MasterMind* mit folgenden Methoden:

```
class MasterMind {

  // erzeugt per Zufall 4-stellige Zahl mit Ziffern 1 bis 6
  MasterMind()

  // liefert die Anzahl an schwarzen Stiften bezogen auf die
  // uebergebenen Ziffern;
  // das Array hat eine Laenge von 4 und enthaelt nur Ziffern
  // 1 bis 6
  int anzahlSchwarzeStifte(int[] ziffern)

  // liefert die Anzahl an weissen Stiften bezogen auf die
  // uebergebenen Ziffern;
  // das Array hat eine Laenge von 4 und enthaelt nur Ziffern
  // 1 bis 6
  int anzahlWeisseStifte(int[] ziffern)

}
```

Nutzen Sie im Konstruktor die Funktion Math.random(), die zufällig double-Werte zwischen 0 und 1 (ausschließlich) generiert (`double zufall = Math.random();`)

Implementieren Sie mit Hilfe der Klasse MasterMind das MasterMind-Spiel, in dem ein menschlicher Spieler gegen den Computer spielt. Der Computer wählt dabei den Code aus und der Mensch muss ihn gemäß der Spielregeln erraten.

Aufgabe 10:

Hintergrund: Eine Bank führt die ersten sechs Buchungen bei jedem Girokonto kostenlos durch. Für die nächsten zehn Buchungen berechnet sie 30 Cent pro Buchung, und für jede weitere Buchung 20 Cent.

Implementierung: Schauen Sie sich das folgende objektorientierte Java-Programm an. In einer Schleife liest es jeweils eine Anzahl an durchzuführenden Buchungen ein, registriert diese für das Konto, fragt beim Konto die bisher angefallenen Kontoführungsgebühren ab und gibt diese auf dem Bildschirm aus:

```
class OOGebuehren {
	public static void main(String[] args) {
		Girokonto meinKonto = new Girokonto();
		while (true) {
			// Anzahl an neuen Buchungen einlesen
			int buchungen;
			do {
				buchungen = IO.readInt("Neue Anzahl Buchungen: ");
			} while (buchungen < 0);

			// vermerkt weitere Buchungen
			meinKonto.neueBuchungen(buchungen);

			// berechnet die seit Kontoeroeffnung
			// angefallenen Kontogebuehren
```

```
                int gebuehren =
                  meinKonto.berechneKontofuehrungsgebuehren();

                // aktuelle Kontofuehrungsgebuehren ausgeben
                IO.println("Gebuehren = " + gebuehren / 100 + ","
                            + gebuehren % 100
                           + " EUR");
          }
     }
}
```

Aufgabe: Implementieren die dazugehörige Klasse `Girokonto`!

Aufgabe 11:

Schauen Sie sich das folgende objektorientierte Java-Programm an. Es simuliert den Verkauf zweier Produkthändler, einer ist Buchhändler, der andere DVD-Händler. Sie verkaufen jeweils nur ein Produkt (Buch bzw. DVD). Anfangs werden der Buchpreis sowie der DVD-Preis eingelesen. Der Benutzer kann nun in einer Schleife wahlweise Bücher oder DVDs kaufen. Die gewünschten Mengen werden dem jeweiligen Händler mitgeteilt. Nachdem der Benutzer den Kaufvorgang abgeschlossen hat, werden die Gesamteinnahmen der beiden Händler auf den Bildschirm ausgegeben.

```
public class UE23Aufgabe11 {

     public static void main(String[] args) {
          double buchPreis = IO.readDouble("Buchpreis: ");
          double dvdPreis = IO.readDouble("DVD-Preis: ");

          Haendler buchHaendler = new Haendler(buchPreis);
          Haendler dvdHaendler = new Haendler(dvdPreis);

          char weiter = 0;
          do {
               char auswahl = IO.readChar("Buch oder DVD kaufen
(b/d)?");
               int anzahl = IO.readInt("Anzahl Produkte: ");
               if (auswahl == 'b') {
                    buchHaendler.kaufen(anzahl);
               } else {
                    dvdHaendler.kaufen(anzahl);
               }

               weiter = IO.readChar("weiter einkaufen(j/n): ");
          } while (weiter == 'j');

          double einnamen = buchHaendler.liefereEinnahmen();
          System.out.println("Einnahmen des Buchhaendlers = " +
einnamen);

          einnamen = dvdHaendler.liefereEinnahmen();
          System.out.println("Einnahmen des DVD-Haendlers = " +
einnamen);
     }

}
```

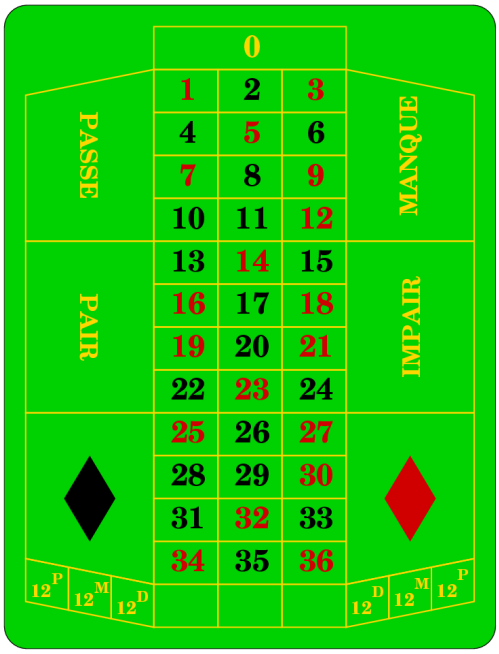
Aufgabe: Implementieren die dazugehörige Klasse `Haendler`!

Beispiel für Programmablauf (Eingaben in <>):

```
Buchpreis: <26.90>
DVD-Preis: <12.00>
Buch oder DVD kaufen (b/d)?<b>
Anzahl Produkte: <2>
weiter einkaufen(j/n): <j>
Buch oder DVD kaufen (b/d)?<d>
Anzahl Produkte: <3>
weiter einkaufen(j/n): <j>
Buch oder DVD kaufen (b/d)?<b>
Anzahl Produkte: <4>
weiter einkaufen(j/n): <n>
Einnahmen des Buchhaendlers = 161.39999999999998
Einnahmen des DVD-Haendlers = 36.0
```

Aufgabe 12:

Definieren Sie eine Klasse `RouletteRad` mit einer Methode `drehen`. Diese soll eine zufällige Zahl zwischen 0 und 36 ermitteln.



(aus Wikipedia)

Für die zuletzt erdrehte Zahl sollen abgefragt werden können: die Zahl, Passe, Manque, Pair, Impair, Red, Black. Sehen Sie geeignete Methoden vor. Schreiben Sie ein kleines Testprogramm.