Programmierkurs Java

Dr.-Ing. Dietrich Boles

Aufgaben zu UE 16 – JDK

(Stand 22.03.2019)

Hinweis:

Die Dokumentation der Klassen der JDK-Klassenbibliothek finden Sie unter: http://docs.oracle.com/javase/<version>/docs/api/ (<version> bitte durch die jeweilige Java-Version ersetzen (4, 5, 6, 7, …)).

Aufgabe 1:

Bei dieser Aufgabe sollen Sie einen kleinen Ausschnitt aus einem Kartenspiel implementieren. Karten in diesem Spiel haben den Typ „kreuz“, „pik“, „herz“ und „karo“ und von jedem Typ gibt es die Kartenwerte „ass“, „2“, „3“, „4“, „5“, „6“, „7“, „8“, „9“, „10“, „bauer“, „dame“, „koenig“.

Für das Kartenspiel benötigen Sie eine Mischmaschine, die die Karten mischen kann. Objekte einer entsprechenden Klasse `MischMaschine` müssen in der Lage sein:

- Karten aufzunehmen,
- die enthaltenen Karten zu mischen,
- alle enthaltenen Karten zu durchlaufen.

Sie kennen die Klasse `java.util.ArrayList` und wissen, dass diese bereits die Anforderungen (1) und (3) über die beide Methoden `add` und die Nutzung der for-each-Schleife erfüllt. Also können Sie zur Realisierung der Klasse `MischMaschine` die Klasse `java.util.ArrayList` und das Konzept der Vererbung nutzen und müssen lediglich eine Methode `mischen`, die das Mischen realisiert, hinzufügen.

Aufgabe (a): Implementieren Sie zunächst eine Klasse `Karte`, die die Karten des Kartenspiels repräsentieren kann. Implementieren Sie weiterhin eine Klasse `MischMaschine` durch Ableitung von der Klasse `java.util.ArrayList`, die Objekte der Klasse `Karte` (ausschließlich) aufnehmen, die enthaltenen Karten mischen und alle enthaltenen Karten durchlaufen kann.

Aufgabe (b): Schreiben Sie ein Programm, in dem alle Karten des Kartenspiels erzeugt und einer Mischmaschine übergeben werden. Die Mischmaschine soll die Karten mischen und anschließend sollen die Karten in der gemischten Reihenfolge auf den Bildschirm ausgeben werden.

Aufgabe 2:

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

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

- Eine Methode `einfuegen` mit zwei `String`-Parametern, nämlich einem Begriff und einer Übersetzung. Ist der Begriff bereits im Wörterbuch vorhanden, soll die alte Übersetzung durch die neue Übersetzung ersetzt werden.
- Eine Methode `getUebersetzung`, die einen String als Parameter übergeben bekommt und als String die im Wörterbuch gespeicherte Übersetzung dieses Begriffs liefert.

Nutzen Sie zur Realisierung der internen Datenstruktur die Klasse `java.util.HashMap`.

Aufgabe (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 3:

Gegenüber Aufgabe 2 ist ein Wörterbuch im Sinne dieser Aufgabe dadurch charakterisiert, dass es mehrere Begriffe enthält, denen jeweils ein oder mehrere Übersetzungen zugeordnet sind.

Aufgabe (a): Implementieren Sie eine Klasse `WoerterbuchExt` mit folgenden Methoden:

- Eine Methode `einfuegen` mit zwei `String`-Parametern, nämlich einem Begriff und einer (u. U. weiteren) Übersetzung.
- Eine Methode `getUebersetzungen`, die einen String als Parameter übergeben bekommt und als `ArrayList<String>` die im Wörterbuch gespeicherten Übersetzungen dieses Begriffs liefert.

Nutzen Sie zur Realisierung der internen Datenstruktur die Klasse `java.util.HashMap`

Aufgabe (b): Schreiben Sie mit Hilfe der Klasse `WoerterbuchExt` 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 4:

Schreiben Sie folgende Programme, indem Sie existierende Klassen des JDK nutzen:

(1) Schreiben Sie ein Programm, das berechnet, wie viele Tage Sie aktuell alt sind (siehe Klassen `java.util.Calendar` und `java.util.Date`).

(2) Schreiben Sie ein Programm, das die Ziehung der Lottozahlen (6 Zahlen zwischen 1 und 49) realisiert (siehe Klasse `java.util.Random`).

(3) Schreiben Sie ein Programm, das vier int-Werte einliest, sie auf einen Stack legt und anschließend in der umgekehrten Reihenfolge wieder auf den Bildschirm ausgibt (siehe Klasse `java.util.Stack`).

(4) Schreiben Sie ein Programm, das einen double-Wert vom Benutzer einliest und den Sinus, Cosinus und Tangens sowie die Quadratwurzel auf dem Bildschirm ausgibt (siehe Klasse `java.lang.Math`)

Aufgabe 5:

Implementieren Sie mit Hilfe der Klasse `java.util.ArrayList` die folgende Klasse `MyStack`.

```
/**
 * A Last-In-First-Out(LIFO) stack of objects.
 */
class MyStack extends ArrayList<Integer> {
    /**
     * Pushes an item onto the stack.
     * @param item the item to be pushed on.
     */
    void push(int item);

    /**
     * Pops an item off the stack.
     */
    int pop();

    /**
     * Peeks at the top of the stack.
     */
    int peek();

    /**
     * Returns true if the stack is empty.
     */
    boolean empty();

    /**
     * Sees if an object is on the stack.
     * @param o the desired object
     * @return the distance from the top, or -1 if it is not found.
     */
    int search(int o);
}
```

Aufgabe 6:

Bei dieser Aufgabe geht es um die Definition einer Klasse *Kalender* zur Terminverwaltung. Die Klasse soll eine Methode *neuerTermin* definieren, über die zu einem bestimmten Datum ein bestimmtes Ereignis in einem Kalender gespeichert werden kann. Außerdem soll sie eine Methode *liefereEreignisse* definieren, die zu einem gewünschten Datum alle im Kalender gespeicherten Ereignisse liefert. Die Repräsentation von Datum und Ereignis erfolgt als Strings.

Ein Testprogramm für die Klasse *Kalender* könnte so aussehen:

```
public static void main(String[] args) {
      Kalender kalender = new Kalender();
      kalender.neuerTermin("03.09.2007", "Geburtstag Oma");
      kalender.neuerTermin("01.01.2007", "Ausschlafen");
      kalender.neuerTermin("03.09.2007", "Schulfest");
      // ...
      String[] ereignisse =
         kalender.liefereEreignisse("03.09.2007");
      for (int i = 0; i < ereignisse.length; i++) {
            System.out.println(ereignisse[i]);
      }
}
```

Das Programm liefert die Ausgabe:

```
Geburtstag Oma
Schulfest
```

Aufgabe: Definieren Sie eine Klasse *Kalender* entsprechend der oben beschriebenen Anforderungen. Nutzen Sie zum Speichern die Klasse `java.util.ArrayList`.

Aufgabe 7:

Schreiben Sie ein Java-Programm, das den DOS-Befehl „dir“ simuliert. Mit diesem Befehl werden bestimmte Informationen aller Elemente eines Verzeichnisses (Dateien und Verzeichnisse) auf dem Bildschirm angezeigt. Der Verzeichnisname soll dabei dem Java-Interpreter beim Programmaufruf übergeben werden.

Die Ausgabe soll für jede Datei bzw. jedes Verzeichnis in einer separaten Zeile erfolgen und folgendes Format haben:

- Datum der letzten Änderung
- „<DIR>“ falls es sich um ein Verzeichnis handelt, ansonsten 5 Leerzeichen
- „r“, falls das Element lesbar ist, ansonsten „-“
- „w“, falls das Element schreibbar ist, ansonsten „-“
- Anzahl an Bytes
- Elementname

Nutzen Sie die JDK-Klasse `java.io.File`

Beispielausgabe:

```
Wed Apr 04 11:32:02 CEST 2012 <DIR>  rw  0   C:\Users\dibo\.android
Wed Feb 16 09:57:56 CET 2011 <DIR>   rw  0   C:\Users\dibo\.appinventor
Fri Apr 01 11:54:28 CEST 2011        rw  155 C:\Users\dibo\.appletviewer
Tue Feb 28 15:37:54 CET 2012 <DIR>   rw  0   C:\Users\dibo\.argouml
Wed Mar 16 14:39:40 CET 2011         rw  96  C:\Users\dibo\.asadminpass
Fri May 07 16:35:29 CEST 2010 <DIR>  rw  0   C:\Users\dibo\.jalopy.15
Mon Nov 29 16:57:28 CET 2010 <DIR>   r-  0   C:\Users\dibo\.jeliot
...
```

Aufgabe 8:

In einem Kreis stehen *n* Kinder (durchnumeriert von 1 bis *n*). Mit Hilfe eines *m*-silbigen Abzählreims wird das jeweils *m*-te unter den noch im Kreis befindlichen Kindern ausgeschieden, bis kein Kind mehr im Kreis steht.

Schreiben Sie ein Java-Programm, das nach Vorgabe von *n* (positive Zahl) und *m* (positive Zahl) die Nummern der Kinder in der Reihenfolge ihres Ausscheidens angibt.

Beispiel*:* Für *n*=6 und *m*=5 ergibt sich die Folge 5, 4, 6, 2, 3, 1.

Hinweis: Nutzen Sie die Klasse `java.util.ArrayList`

Aufgabe 9:

Mit der **Java Reflection API** kann man zur Laufzeit Informationen über Objekte und Klassen erhalten. Genau das sollen Sie in dieser Aufgabe auch tun. Sie benötigen dafür folgende Klassen des API:

```
package java.lang;
public class Object {

    // zu jeder Klasse gibt es in Java ein so genanntes
    // Klassenobjekt; dieses kann man durch den Aufruf
    // dieser Methode für ein Objekt der Klasse erfragen;
    // durch Aufruf von (new String()).getClass() lässt sich
    // bspw. das Klassenobjekt der Klasse String ermitteln
    public java.lang.Class getClass()
}

package java.lang;
public class Class { // repräsentiert eine Klasse

    // liefert den vollständigen Namen der Klasse
    public String getName()

    // liefert alle Attribute der Klasse
    public java.lang.reflect.Field[] getDeclaredFields()
}

package java.lang.reflect;
public class Field { // repräsentiert ein Attribut

    // liefert eine Kodierung aller Modifier
    // (public, static, final, ...) eines Attributs;
    // übergibt man den Code der Methode Modifier.toString,
    // erhält man eine passende Stringrepräsentation
    public int getModifiers()

    // jeder Typ wird in Java durch eine Klasse
    // repräsentiert; diese Methode liefert das Klassenobjekt
    // des Typs des Attributs; ist das
    // Attribut bspw. vom Typ int, wird das Klassenobjekt
    // des Standardtyps int geliefert
    public java.lang.Class getType()

    // liefert den Namen eines Attributes
    public String getName()
}
```

Aufgabe: Implementieren Sie auf der Grundlage der Java Reflection API eine Funktion

```
static void printClass(Object obj)
```

die die Klassendefinition (eingeschränkt auf Klassenname und Attribute (jeweils Modifier, Typ und Name)) der Klasse von obj auf den Bildschirm ausgibt.

Beispiele:

Gegeben sei die folgende Klasse

```
class CX {
      int i;
      static int s;
      final public static int k = 4711;
}
```

Ein Aufruf der Funktion `printClass` mittels

```
printClass(new CX());
```

sollte dann folgende Bildschirmausgabe produzieren:

```
class CX {
  int i;
  static int s;
  public static final int k;
}
```

Ein Aufruf mittels

```
printClass(new java.lang.Boolean(false));
```

sollte folgende Ausgabe liefern:

```
class java.lang.Boolean {
  public static final java.lang.Boolean TRUE;
  public static final java.lang.Boolean FALSE;
  public static final java.lang.Class TYPE;
  private final boolean value;
  private static final long serialVersionUID;
}
```

Aufgabe 10:

In nahezu jedem Lehrbuch finden Sie im hinteren Teil so genannte Indexe bzw. Sachwortverzeichnisse. Hier werden wichtige Begriffe, die im Buch vorkommen, sowie die jeweiligen Seitenzahlen, auf denen die Begriffe vorkommen, aufgelistet.

Implementieren Sie eine ADT-Klasse `Index`, die einen solchen Index repräsentiert. Die Klasse soll folgende Methoden zur Verfügung stellen:

- Einen Default-Konstruktor, der einen leeren Index erstellt.
- Einen Copy-Konstruktor; der neue Index soll dabei alle Einträge enthalten, die der als Parameter übergebene Index zum Zeitpunkt des Aufrufs enthält.
- Eine Methode `clone` zum Klonieren eines Index (Überschreiben der von der Klasse `Object` geerbten Methode `clone`). Der neue Index soll dabei alle

Einträge enthalten, die der aufgerufene Index zum Zeitpunkt des Aufrufs enthält.

- Eine Methode `hinzufuegen`, die einen Begriff (String) sowie eine Seitenzahl (int-Wert) als Parameter übergeben bekommt und die den Begriff mit der Seitenzahl im Index vermerkt. Dabei gilt: Begriffe dürfen nicht mehrfach im Index vorkommen und zu jedem Begriff darf eine Seitenzahl höchstens einmal im Index vorkommen.
- Eine Methode `toString`, die eine String-Repräsentation des aktuellen Index liefert (Überschreiben der von der Klasse `Object` geerbten Methode `toString`). Die String-Repräsentation soll dabei folgendermaßen aufgebaut sein: Für jeden im Index vermerkten Begriff soll der String jeweils eine Zeile der folgenden Form enthalten: `<begriff>: <seitenzahl 1> <seitenzahl 2> … <seitenzahl n>` (also bspw.: `Hamster: 2 45 123`)

Zusatz: Geben sie sowohl die Einträge des Index als auch die Seitenzahlen sortiert aus (lexikographische bzw. natürliche Ordnung der Begriffe bzw. der Seitenzahlen). Suchen Sie im JDK nach geeigneten Klassen bzw. Methoden zum Sortieren.

Testen: Ob Ihre Klasse (zumindest teilweise) korrekt ist, können Sie testen, indem Sie das folgende Testprogramm ausführen:

```java
public class UE16Aufgabe10 {

    public static void main(String[] args) {
        Index index = new Index();

        index.hinzufuegen("Objekt", 1);
        index.hinzufuegen("Objekt", 15);
        index.hinzufuegen("Objekt", 3);
        index.hinzufuegen("Objekt", 15);

        Index index2 = new Index(index);

        index.hinzufuegen("Hamster", 45);
        index.hinzufuegen("Hamster", 2);
        index.hinzufuegen("Hamster", 199);
        index.hinzufuegen("Hamster", 45);

        System.out.println(index);
        System.out.println(index2);

    }
}
```

Es sollte folgende Ausgabe (unsortiert) erscheinen:

```
Objekt: 1 15 3
Hamster: 45 2 199

Objekt: 1 15 3
```

Aufgabe 11:

Damit man sich Passwörter gut merken kann, ohne dass sich diese leicht erraten lassen, kann man Passwörter aus Merksätzen bilden. Wir wollen nach folgenden Regeln Passwörter aus Sätzen bilden:

- Das Passwort wird als Folge der letzten Zeichen eines jeden Wortes im Satz gebildet.
- Wörter des Satzes sollen alle durch beliebig viele Leerzeichen abgetrennnten Teilfolgen sein. Werden Satzzeichen direkt hinter ein Wort geschrieben, so gehören sie zu dem Wort.
- Bei Buchstaben des Unicode-Alphabets soll alternierend mal ein Gross- dann wieder ein Kleinbuchstabe verwendet werden. Der erste Buchstabe soll großgeschrieben werden.

Beispiel: Der Satz "Wenn es regnet, dann wird es nass ." besteht aus den Teilwörtern "Wenn" "es" "regnet," "dann" "wird" "es" "nass" "." und das dazugehörige Passwort ist "Ns,NdSs."

Aufgabe: Implementieren Sie eine Java-Klasse `PasswortGenerator` mit einer Klassenmethode, die zu einem als String-Parameter übergebenen Satz ein Passwort nach obiger Methode ermittelt und als String zurückgibt.

Verwenden Sie zur Erledigung der einzelnen anfallenden Teilaufgaben falls möglich Methoden der Klassen `java.lang.String` und `java.lang.Character`.

Aufgabe 12:

Sokoban ist ein bekanntes Computerspiel. Es wird auf einem rechteckigen Spielfeld gespielt, das aus einzelnen gleich-großen Feldern besteht. Auf den Feldern können sich Objekte befinden, von denen es folgende gibt:

- Sokoban: die Spielfigur. Es existiert immer genau ein Sokoban.
- Kisten: Kisten können vom Sokoban verschoben werden.
- Zielfelder: Es existieren immer genauso viele Zielfelder wie Kisten. Ziel des Spiels ist es, alle Kisten durch den Sokoban auf jeweils ein Zielfeld zu verschieben.
- Wände: Durch Wände können Wege blockiert sein. Das Spielfeld ist immer komplett von Wänden umgeben.

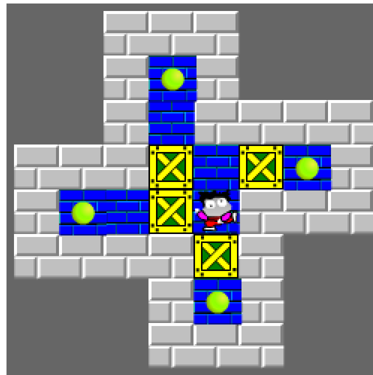
Auf jedem Feld kann maximal ein Sokoban, eine Kiste, eine Wand oder ein Zielfeld vorhanden sein. Auf einem Zielfeld kann zusätzlich noch eine Kiste oder der Sokoban stehen.

In jedem Spielzug kann der Sokoban durch den Spieler jeweils um ein Feld in eine der Richtungen Nord, Süd, West oder Ost verschoben werden, falls sich in Zugrichtung vor dem Sokoban keine Wand befindet. Steht in Zugrichtung unmittelbar vor dem Sokoban eine Kiste, so wird diese ebenfalls um ein Feld in dieselbe Richtung verschoben. Ein Verschieben einer Kiste ist jedoch nicht möglich, wenn sich hinter der Kiste eine weitere Kiste oder eine Wand befindet.

Das Spiel ist beendet, wenn jede Kiste auf einem Zielfeld steht.

Viele Sokoban-Implementierungen verwenden zur Beschreibung der Ausgangslevel ein einfaches ASCII-Format. Dabei wird eine Wand durch das Symbol „#“ dargestellt, der Sokoban durch ein „@“, ein Zielfeld als „.“ und eine Kiste als „$“. Steht eine Kiste auf einem Zielfeld, wird dies durch ein „*“ und steht der Sokoban auf einem Zielfeld wird dies durch ein „+“ repräsentiert. Beispiel (links ASCII, rechts graphisch):

```
  ###
  #.#
  # ####
###$ $.#
#. $@###
####$#
   #.#
   ###
```



Vorgabe:

Im Folgenden wird ein Ausschnitt aus einer Implementierung des Sokoban-Spiels gegeben:

```
// existierende Sokoban-Objekte
enum SokobanObjekt { Sokoban, Kiste, Wand, Zielfeld }

// auf einem Feld koennen sich ein oder mehrere SokobanObjekte befinden
class FeldBelegung extends java.util.ArrayList<SokobanObjekt> { }

public class Sokoban {
     private FeldBelegung[][] feld; // aktuelle Belegung aller Felder

     // Voraussetzung: die Datei enthaelt eine korrekte ASCII-Repraesentation
     // eines Sokoban-Spielfeldes
     public Sokoban(String dateiname) {
           char[][] feld = IO.readFileAsCharMatrix(dateiname);
           this.feld = convert(feld);
     }

public void spielen() {
           this.print();
           while (!spielEnde()) {
                 char eingabe = IO.readChar("N/O/W/S?");
                 switch (eingabe) {
                 case 'N':   case 'n':   nachNorden(); break;
                 case 'O':   case 'o':   nachOsten();  break;
                 case 'W':   case 'w':   nachWesten(); break;
                 case 'S':   case 's':   nachSueden(); break;
                 }
                 this.print();
           }
           System.out.println("Geschafft!");
     }
```

```
	// konvertiert ein Sokoban-Spielfeld aus dem ASCII-Format in eine
interne
	// Repraesentation
	// Achtung: Es wird vorausgesetzt, dass die ASCII-Beschreibung
korrekt ist!
	static FeldBelegung[][] convert(char[][] feld) {
		FeldBelegung[][] sokobanFeld =
		 new FeldBelegung[feld.length][feld[0].length];
		for (int r = 0; r < feld.length; r++) {
			for (int s = 0; s < feld[0].length; s++) {
				sokobanFeld[r][s] = new FeldBelegung();
				switch (feld[r][s]) {
				case '#':
					sokobanFeld[r][s].add(SokobanObjekt.Wand);
					break;
				case '@':
					sokobanFeld[r][s].add(SokobanObjekt.Sokoban);
					break;
				case '.':
					sokobanFeld[r][s].add(SokobanObjekt.Zielfeld);
					break;
				case '$':
					sokobanFeld[r][s].add(SokobanObjekt.Kiste);
					break;
				case '*':
					sokobanFeld[r][s].add(SokobanObjekt.Kiste);
					sokobanFeld[r][s].add(SokobanObjekt.Zielfeld);
					break;
				case '+':
					sokobanFeld[r][s].add(SokobanObjekt.Sokoban);
					sokobanFeld[r][s].add(SokobanObjekt.Zielfeld);
					break;
				}
			}
		}
		return sokobanFeld;
	}


	void print() { /* Implementierung hier unwichtig */  }

	boolean spielEnde() { /* Implementierung: Ihre Aufgabe */ }

	void nachNorden() { /* Implementierung: Ihre Aufgabe */ }

	void nachSueden() { /* Implementierung hier unwichtig */   }

	void nachWesten() { /* Implementierung hier unwichtig */   }

	void nachOsten() { /* Implementierung hier unwichtig */    }


	public static void main(String[] args) {
		Sokoban sokoban = new Sokoban("bsp2.soko");
		sokoban.spielen();
	}
}
```

Aufgabe:

Implementieren Sie die fehlenden Methoden `spielEnde` und `nachNorden`, und zwar auf der Grundlage der im Konstruktor erstellten Datenstruktur (`private`

`FeldBelegung[][] feld;`), d.h. sie dürfen die Datenstruktur nicht in eine andere Datenstruktur überführen.

- `spielEnde`: Überprüft das Spielende von Sokoban (siehe obige Spielregeln).
- `nachNorden`: Ausführung eines Spielzugs, bei dem versucht wird, den Sokoban nach Norden zu verschieben (siehe obige Spielregeln), d.h. die Datenstruktur muss entsprechend untersucht und ggfls. verändert werden.

Das vorgegebene Programm dürfen Sie nicht ändern, sondern nur ergänzen.

Aufgabe 13:

Eine Balkenwaage ist eine Wägevorrichtung, die aus einem waagerechten Balken besteht, der beweglich an einer waagerechten Achse gelagert ist. An jedem Balkenende befindet sich eine Waagschale (Wikipedia). Auf die Waagschalen können Körper mit einem Gewicht gelegt, ihre Gewichte gemessen und verglichen werden.

Gegeben sei folgende Java-Klasse zur Repräsentation von Körpern:

```
public class Koerper {
     private int gewicht;

     public Koerper(int gewicht) {
          this.gewicht = gewicht > 0 ? gewicht : 1;
     }

     public int getGewicht() {
          return this.gewicht;
     }

     public void setGewicht(int gewicht) {
          this.gewicht = gewicht;
     }
}
```

Definieren Sie eine Klasse *Balkenwaage*. *Balkenwaage*-Objekte sollen eine linke und eine rechte Waagschale besitzen, auf die jeweils prinzipiell beliebig viele *Koerper*-Objekte gelegt werden können. ACHTUNG: Ein und dasselbe *Koerper*-Objekt darf dabei natürlich zu jedem Zeitpunkt maximal einmal auf der Balkenwaage liegen!

Überlegen Sie sich zunächst eine geeignete Datenstruktur und definieren Sie entsprechende Attribute. Definieren und implementieren Sie dann folgende Methoden:

- Einen Konstruktor, der eine leere Balkenwaage erzeugt/initialisiert.
- Eine Methode *hinzufuegenLinks*, mit der ein als Parameter übergebenes *Koerper*-Objekt der linken Waagschale der Balkenwaage hinzugefügt wird.
- Eine Methode *hinzufuegenRechts*, mit der ein als Parameter übergebenes *Koerper*-Objekt der rechten Waagschale der Balkenwaage hinzugefügt wird.
- Eine Methode *entfernen*, mit der ein als Parameter übergebenes *Koerper*-Objekt von der Waage entfernt wird.
- Eine boolesche Methode *istInWaage*, die genau dann *true* liefert, wenn die Summe der Gewichte der *Koerper*-Objekte auf den beiden Waagschalen der Balkenwaage identisch ist.
- Eine boolesche Methode *schwererLinks*, die genau dann *true* liefert, wenn die Summe der Gewichte der *Koerper*-Objekte auf der linken Waagschale größer ist als die Summe der Gewichte der *Koerper*-Objekte auf der rechten Waagschale der Balkenwaage.
- Eine Methode *toString* (Überschreiben der Methode *toString* der Klasse *Object*), die einen String im Format <Gewicht links>:<Gewicht rechts> liefert, in dem die Gesamtgewichte der *Koerper*-Objekte auf der jeweils linken und rechten Waagschale der Balkenwaage aufgeführt werden (bspw. „20:30“ wenn das Gesamtgewicht der linken Waagschale 20 und der rechten Waagschale 30 Einheiten beträgt).

Das folgende Beispielprogramm demonstriert den Aufruf der entsprechenden Methoden:

```
Balkenwaage waage = new Balkenwaage();
Koerper k1 = new Koerper(10);
Koerper k2 = new Koerper(5);
waage.hinzufuegenLinks(k1);
waage.hinzufuegenRechts(k2);
while (waage.schwererLinks()) {
      waage.hinzufuegenRechts(new Koerper(1));
}
System.out.println(waage.isInWaage());
waage.hinzufuegenRechts(k1);
waage.entfernen(k2);
System.out.println(waage.toString());
```

Aufgabe 14:

Ein Ausdruck kann aus Zahlen und Operatoren bestehen. Er berechnet und liefert einen Wert. In dieser Aufgabe wird ein Ausdruck durch ein char-Array dargestellt, wobei als gültige char-Zeichen die Ziffern (0, 1, …, 9) sowie die zweistelligen Operatoren +, -, * und / gelten. Multiplikation und Division haben dabei dieselbe Priorität und Addition und Subtraktion haben dieselbe Priorität. Die Priorität von Multiplikation und Division ist höher als die Priorität von Addition und Subtraktion. Die

Operatoren sind linksassoziativ. Zahlen setzen sich aus mehreren aufeinander folgenden Ziffern zusammen.

Ein „gültiger Ausdruck“ im Sinne dieser Aufgabe beginnt immer mit einer Zahl und endet mit einer Zahl. Zwischen zwei Zahlen steht immer genau ein Operator.

Aufgabe: Implementieren Sie in Java eine Seiteneffekt-freie Funktion

```
static int berechne(char[] ausdruck)
```

der ein gültiger Ausdruck als char-Array übergeben wird und die daraufhin den Wert des Ausdrucks berechnet und als Funktionswert liefert. Sie können davon ausgehen, dass gültige Ausdrücke übergeben werden (bspw. keine Zahlen, die nicht mehr als int-Wert repräsentierbar sind) und dass eine fehlerfreie Auswertung der Ausdrücke möglich ist (bspw. keine Division durch 0).

Test: Das folgende kleine Testprogramm sollte die Ausgabe

```
113+23=136
28-10/5+7*6=68
```

erzeugen:

```
public static void main(String[] args) {
     char[] ausdruck1 = {'1','1','3','+','2','3'};
     System.out.print(ausdruck1);
     System.out.println("=" + berechne(ausdruck1));
     char[] ausdr2 = {'2','8','-','1','0','/','5','+','7','*','6'};
     System.out.print(ausdr2);
     System.out.println("=" + berechne(ausdr2));
}
```

Algorithmus: Folgender der Funktion *berechne* zugrunde liegender Algorithmus ist dabei vorgegeben und zu implementieren:

(1) Durchlaufe den übergebenen Ausdruck (char-Array) von links nach rechts, identifiziere dabei Zahlen und Operatoren und speichere alle identifizierten Zahlen hintereinander in einer Zahlen-ArrayList und alle identifizierten Operatoren hintereinander in einer Operatoren-ArrayList.

(2) Durchlaufe die Operatoren-ArrayList von vorne nach hinten und suche zunächst nach den höherwertigen Operatoren * und /. Beim Auffinden eines höherwertigen Operators am Index i wende die entsprechende Operation auf die beiden Zahlen am Index i und i+1 der Zahlen-ArrayList an, setze die Zahl am Index i der Zahlen-ArrayList auf das Ergebnis der Operation und entferne die Zahl am Index i+1 aus der Zahlen-ArrayList. Entferne weiterhin auch den Operator am Index i aus der Operatoren-ArrayList.

(3) Führe denselben Teilalgorithmus wie in (2) nun für die minderwertigen Operatoren + und - durch.

(4) Liefere als Ergebnis der Berechnung die Zahl am Index 0 der Zahlen-ArrayList.

Beispielhafte Datenstrukturen: (für den zweiten Ausdruck von den Test-Beispielen oben):

Nach (1):	Zahlen-ArrayList:	28, 10, 5, 7, 6
	Operatoren-ArrayList:	-, /, +, *
Nach (2a):	Zahlen-ArrayList:	28, 2, 7, 6
	Operatoren-ArrayList:	-, +, *
Nach (2b):	Zahlen-ArrayList:	28, 2, 42
	Operatoren-ArrayList:	-, +
Nach (3a):	Zahlen-ArrayList:	26, 42
	Operatoren-ArrayList:	+
Nach (3b):	Zahlen-ArrayList:	68
	Operatoren-ArrayList:	(leer)

Aufgabe 15:

Mit der Java Reflection API kann man zur Laufzeit Informationen über Objekte und Klassen erhalten und Methoden der Objekte ausführen. Genau das sollen Sie in dieser Aufgabe auch tun. Sie benötigen dafür folgende Klassen des API:

```
package java.lang;
public class Object {
    // zu jeder Klasse gibt es in Java ein so genanntes
    // Klassenobjekt; dieses kann man durch den Aufruf
    // dieser Methode für ein Objekt der Klasse erfragen;
    // durch Aufruf von (new String()).getClass() lässt sich
    // bspw. das Klassenobjekt der Klasse String ermitteln
    public java.lang.Class getClass()
}

package java.lang;
public class Class { // repräsentiert eine Klasse

    // liefert den vollständigen Klassennamen der Klasse des
Klassenobjektes
    public String getName()

    // liefert alle Methoden der Klasse des Klassenobjektes
    public java.lang.reflect.Method[] getDeclaredMethods()
}

package java.lang.reflect;
public class Methode { // repräsentiert eine Methode

    // liefert den Namen der Methode
    public String getName()

    // liefert die Anzahl an Parametern der Methode
    public int getParameterCount()

    // jeder Typ wird in Java durch eine Klasse
    // repräsentiert; diese Methode liefert das Klassenobjekt
    // des Typs des Rückgabewertes der Methode; ist der
    // Rückgabetyp der Methode bspw. String, wird das Klassenobjekt
```

```
        // der Klasse java.lang.String geliefert
        public java.lang.Class getReturnType()

        // führt eine parameterlose Methode für das als Parameter
        // übergebene Objekt aus und liefert den Rückgabewert der ausgeführten
        // Methode; für eine Methode namens f also äquivalent zu obj.f();
        public Object invoke(Object obj)

    }
```

Aufgabe: Implementieren Sie auf der Grundlage der Java Reflection API eine Funktion

```
static void callStringMethods(Object obj)
```

die für alle Parameter-losen Methoden mit dem Rückgabetyp String eines beliebigen Objektes *obj* eine Zeile folgender Form auf die Konsole ausgibt:

Die Methode <name> liefert: <wert>

Wobei <name> für den jeweiligen Methodennamen steht und <wert> für den durch die Methode zurückgegebenen Wert.

Beispiele: Gegeben sei die folgende Klasse

```
class X {
  public String f() {
    return "X::f";
  }

  public int g() {
    System.out.println("X::g");
    return 1;
  }

  public String h() {
    return "X::h";
  }
}
```

Ein Aufruf von **callStringMethods(new X());** sollte dann folgende Ausgabe produzieren:

```
Die Methode f liefert: X::f
Die Methode h liefert: X::h
```

Aufgabe 16:

Sie kennen aus der Vorlesung Warteschlangen bzw. Queues. Queues sind Datenstrukturen, die nach dem FIFO-Prinzip (First-In-First-Out) arbeiten. Über eine Methode **enqueue** kann ein neues Element am Schwanz der Queue eingetragen werden. Mit Hilfe einer Methode **dequeue** wird das älteste Element am Kopf der Queue geliefert und aus der Queue entfernt.

In dieser Aufgabe geht es um so genannte ***TeamQueues***. Elemente einer TeamQueue sind Personen, die jeweils einem Team angehören. Wenn eine neue

Person in eine TeamQueue eingetragen werden soll, wird zunächst die Queue von Kopf bis Schwanz untersucht, ob bereits ein oder mehrere Teamkameraden (d.h. Elemente desselben Teams) in der Queue sind. Falls das der Fall ist, wird die Person unmittelbar hinter diesen (d.h. als letzte des Teams) in die Queue eingefügt. Falls nicht, wird das Element ganz hinten an die Queue angehängt. Das Dequeuing aus einer TeamQueue erfolgt analog zum Dequeueing aus einer normalen Queue.

Gegeben seien folgende Klassen **`Team`**, **`Person`** und **`Queue`**. Implementieren Sie eine solche Klasse **`TeamQueue`**, indem sie diese von der Klasse **`Queue`** ableiten und ausschließlich die erforderlichen Methode(n) adäquat überschreiben.

```
class Team {
	private String name;

	public Team(String name) { this.name = name; }
}

class Person {
	private String name;
	private Team team;

	public Person(String name, Team team) {
		this.name = name; this.team = team;
	}

	public String getName() { return name; }
	public Team getTeam() { return team; }
}

class Queue {
	protected ArrayList<Person> persons;

	public Queue() {
		persons = new ArrayList<Person>();
	}

	public void enqueue(Person p) {
		persons.add(p);  // fügt hinten an
	}

	public Person dequeue() {
		return persons.remove(0);
```

```
        // entfernt und liefert vorderes Element
    }

    public boolean isEmpty() {
        return this.persons.isEmpty();
    }
}
```

Das folgende kleine Testprogramm

```
    Team team1 = new Team("Maenner");
    Team team2 = new Team("Frauen");
    TeamQueue queue = new TeamQueue();
    queue.enqueue(new Person("Karl", team1));
    queue.enqueue(new Person("Maria", team2));
    queue.enqueue(new Person("Otto", team1));
    queue.enqueue(new Person("Paula", team2));
    System.out.println(queue.dequeue().getName());
    System.out.println(queue.dequeue().getName());
    queue.enqueue(new Person("Kevin", team1));
    queue.enqueue(new Person("Karla", team2));
    while (!queue.isEmpty()) {
        System.out.println(queue.dequeue().getName());
    }
```

sollte ausgeben:

```
Karl
Otto
Maria
Paula
Karla
Kevin
```