252-0027-00: Einführung in die Programmierung

Übungsblatt 10 (ohne Bonus)

Abgabe: 5. Dezember 2023, 23:59

Die Bonusaufgabe für diese Übung wird erst am Dienstag Abend der Folgewoche (also am 05.12) um 17:00 Uhr publiziert und Sie haben dann 2 Stunden Zeit, diese Aufgabe zu lösen. Der Abgabetermin für die anderen Aufgaben ist wie gewohnt am Dienstag Abend um 23:59. Bitte planen Sie Ihre Zeit entsprechend.
Checken Sie mit Eclipse wie bisher die neue Übungs-Vorlage aus. Importieren Sie das Eclipse-Projekte für die Aufgaben ohne Bonus. Vergessen Sie nicht, Tests zu schreiben! Auch Ausnahmefälle (Exceptions) können mit JUnit getestet werden, z.B. so:

```
boolean thrown = false;
try {
    // code
} catch (SomeException e) {
    thrown = true;
}
if (!thrown) {
    fail("expected some exception");
}
```

Aufgabe 1: Loop Invariante

Gegeben ist eine Postcondition für das folgende Programm

```
public int compute(String s, char c) {
  int x;
  int i;

  x = 0;
  i = -1;

  // Loop Invariante:
  while (x < s.length() && i < 0) {
    if (s.charAt(x) == c) {
      i = x;
```

```
    }
    x = x + 1;
  }

  // Postcondition:
  //   (0 <= i && i < s.length() && s.charAt(i) == c) || count(s, c) == 0
  return i;
}
```

Die Methode `count(String s, char c)` gibt zurück wie oft der Character `c` im String `s` vorkommt. Schreiben Sie die Loop Invariante in die Datei "LoopInvariante.txt". **Achtung:** Die Aufgabe ist schwerer als es zuerst scheint. Überprüfen Sie Ihre Lösung sorgfältig.

Aufgabe 2: Cyclic List (Recap!)

Bisher haben Sie einfach-verkettete Listen und doppelt-verkettete Listen gesehen. Zusätzlich wurde ein `IntList` Interface eingeführt (siehe Anhang A), welches die Methoden beider Arten von Listen abstrahiert.

a) In dieser Aufgabe üben Sie den Umgang mit Interfaces. Die Klasse `LinkedIntList` hat alle Methoden, welche vom Interface `IntList` gefordert werden. Definieren Sie, dass die Klasse `LinkedIntList` das Interface `IntList` implementiert. Implementieren Sie dann eine Methode `ListUtil.addMin(IntList x)`, welche der Liste `x` die kleinste Zahl anhängt, welche in `x` enthalten ist. Implementieren Sie zuletzt die Methode `ListUtil.addMinImpl(LinkedIntList x)`, welche ebenfalls der Liste `x` die kleinste Zahl anhängt, welche in `x` enthalten ist, aber dafür die Methode `ListUtil.addMin` verwenden soll. Sie dürfen für beide Methoden annehmen, dass das Argument mindestens ein Element enthält.

b) In dieser Aufgabe implementieren Sie eine neue Variante einer Liste, die zyklische Liste, welche ebenfalls das `IntList` Interface implementiert. Zyklische Listen sind ähnlich zu einfach-verketteten Listen mit dem Unterschied, dass das `next` Feld des letzten `Nodes` der Liste, falls es einen letzten Knoten gibt, auf den ersten `Node` der Liste zeigt. Die Knoten der Liste bilden also einen Zyklus. Zusätzlich hat die Liste kein Feld für den ersten Knoten der Liste, da dies unnötig ist. Das Feld `last`, das auf den letzten Knoten zeigt, ist nach wie vor vorhanden. Abbildung 1 zeigt eine solche zyklische Listen mit den Elementen 1, 3, 3, 7. Implementieren Sie die zyklische Liste in der Datei "CircularLinkedIntList.java". Einige Tests für die Liste finden Sie in `IntListTest`.

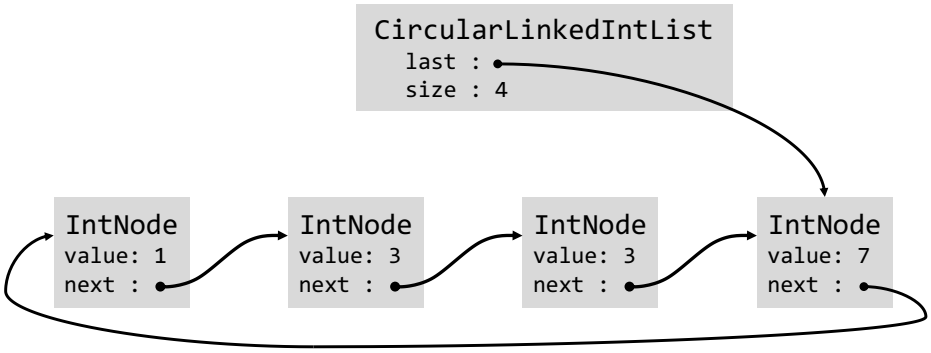


Abbildung 1: Zyklische Liste mit Werten 1, 3, 3, 7.

Aufgabe 3: Rechnungen (Erweitert)

In dieser Aufgabe erweitern Sie eine vorherige Aufgabe, in welcher ein System für Stromverbräuche Rechnungen erstellt. Konkret gibt es drei Erweiterungen: (1) Es sollen auch nicht korrekt formatierte Eingabedateien gehandhabt werden. (2) Ein Kunde kann eine beliebige Anzahl von Verbrauchswerten haben. (3) Es gibt eine neue Unteraufgabe b. In der folgenden Aufgabenbeschreibung für Unteraufgabe a sind die Änderung in **bold** markiert.

a) Vervollständigen Sie die `process`-Methode in der Klasse `Bills`. Die Methode hat zwei Argumente: einen `Scanner`, von dem Sie den Inhalt der Eingabedatei lesen sollen, und einen `PrintStream`, in welchen Sie die unten beschriebenen Informationen schreiben.

   Ihr Programm muss **auch mit manchen nicht korrekt formatierten Eingabedatein umgehen**. **Die Aufgabestellung gibt an, wie mit nicht korrekt formatierten Eingaben umzugehen ist**. Ein Beispiel einer korrekt formatierten Datei finden Sie im Projekt unter dem Namen "Data.txt". Exceptions im Zusammenhang mit Ein- und Ausgabe können Sie ignorieren.

   Eine valide Eingabedatei enthält Zeilen, die entweder den Tarif, der angewendet werden soll, oder die Daten für den Stromverbrauch eines Kunden beschreiben. Der Verbrauch eines Kunden ist niemals grösser als 100000 Kilowattstunden.

   Eine Tarifbeschreibung hat folgendes Format:

$$\texttt{Tarif}\,␣\,n\,␣\,l_1\,␣\,p_1\,\ldots\,l_n\,␣\,p_n$$

   Folgendes gilt für die Parameter:

   - `Tarif` (so geschrieben) ist ein Keyword, das angibt, dass die Zeile einen Tarif beschreibt.
   - n is eine positive ganze Zahl, welche die Anzahl der Intervalle angibt, für welche ein Strompreis festgelegt ist.
   - Auf n folgt eine Folge von n Paaren von ganzen Zahlen ($l_1\,␣\,p_1\,\ldots\,l_n\,␣\,p_n$). Die erste Zahl eines Paares gibt die Obergrenze des Intervalls an und die zweite den Preis für diesen Verbrauch; für ein i, so dass $1 \leq i \leq n$, ist l_i also der Verbrauch (in Kilowattstunden), bis

zu welchem der Strompreis p_i (in Rappen pro Kilowattstunde) zur Anwendung kommt ($l_i > 0$ und $p_i \geq 0$). Die Paare sind jeweils mit einem Whitespace voneinander getrennt (und l_i und p_i jeweils voneinander auch).

Hier sind einige Beispiele für Tarifbeschreibungen:

- `Tarif 1 100000 30`
  Es gibt ein Intervall und für jede Kilowattstunde müssen 30 Rappen bezahlt werden.
- `Tarif 2 1000 10 100000 30`
  Es gibt zwei Intervalle. Die ersten 1000 Kilowattstunden kosten 10 Rappen pro Kilowattstunde. Der Rest kostet 30 Rappen pro Kilowattstunde.
- `Tarif 3 100 40 1000 10 100000 30`
  Es gibt drei Intervalle. Die ersten 100 Kilowattstunden kosten 40 Rappen pro Kilowattstunde. Die nächsten 1000 Kilowattstunden kosten 10 Rappen pro Kilowattstunde. Der Rest kostet 30 Rappen pro Kilowattstunde.

Wenn ein Kunde insgesamt 2000 Kilowattstunden verbraucht, so beträgt die Rechung für das erste Beispiel 600 Franken, im zweiten Beispiel 400 Franken und 410 Franken im dritten.

Die Beschreibung des Stromverbrauchs eines Kunden hat folgendes Format:

$$ID \textvisiblespace v_{q_1} \textvisiblespace v_{q_2} \ldots v_{q_m}$$

Hierbei gilt für die Parameter:

- ID ist eine positive ganze Zahl.
- v_{q_i} **ist eine ganze Zahl, die den Verbrauch im i-ten Quartal in Kilowattstunden angibt ($v_{q_i} \geq 0$). Es kann eine beliebige Anzahl von Verbrauchwerten geben (auch keine).**

Hier ist ein Beispiel für eine Verbrauchbeschreibung:

```
115 0 0 0 10 2000 12
```

Der Kunde mit ID 115 hat im vierten Quartal 10 Kilowattstunden, im füften Quartal 2000 Kilowattstunden, und im sechsten Kilowattstunden 12 Kilowattstunden Strom verbraucht.

Ein einmal gelesener Tarif wird für alle Kunden angewendet, die nach dieser Tarifinformation in der Eingabedatei erscheinen. Wenn ein neuer Tarif erscheint, dann gilt der danach für die weiteren Kunden bis auf Weiteres. Sie können davon ausgehen, dass eine Kunden-ID nur einmal in der Eingabedatei vorkommen kann und dass die erste Zeile der Eingabedatei eine Tarifbeschreibung ist. Implementieren Sie die **Ihr Program soll Zeilen im Input ignorieren, welche weder mit "Tarif" noch einem positiven Integer beginnen. Falls ein Verbrauchswert im Input kein positiver Integer ist, dann soll eine `IllegalFileFormatException` geworfen werden. Die `IllegalFileFormatException` wird zur Verfügung gestellt.**

Die Methode `process` soll die Eingabedatei verarbeiten und für jeden Kunden eine Zeile

$$ID \textvisiblespace b$$

in den der Methode in `output` übergebenen `PrintStream` schreiben. *ID* ist die ID des Kunden (`int`) und *b* ist eine ganze Zahl, die die jeweilige Rechnung für den Gesamtverbrauch **in Franken** angibt. (Zuerst muss der Gesamtverbrauch berechnet werden, dann kann der entsprechende Tarif angewendet werden.) Berechnen Sie den Rechnungsbetrag und runden Sie das Resultat anschliessend (vor der Ausgabe, aber nach den Berechnungen) auf die nächste ganze Zahl. Sie können hierfür die Methode `Math.round(double a)` verwenden. Die Ausgabe darf keine weiteren Zeichen enthalten. Sie können den Betrag so ausgeben, wie er von der `println`-Anweisung herausgegeben wird, d.h. Sie brauchen das Ergebnis nicht zu formatieren.

b) Implementieren Sie zusätzlich die Methode `query(int lowB, int upB)`. Diese Methode wird nur nach einem Aufruf von `process` aufgerufen. Die Methode `query(int lowB, int upB)` gibt die Id des Kunden zurück, der den grössten Rechnungsbetrag im Interval $[lowB, upB]$ hatte. Der Rechnungsbetrag R' dieses Kunden K' erfüllt die Bedingungen

   - `lowB` $\leq R' \leq$ `upB` *und*
   - Es gibt keinen Kunden K'' mit Rechnungsbetrag R''. so dass $R' < R''$ und $R'' \leq$ `upB`

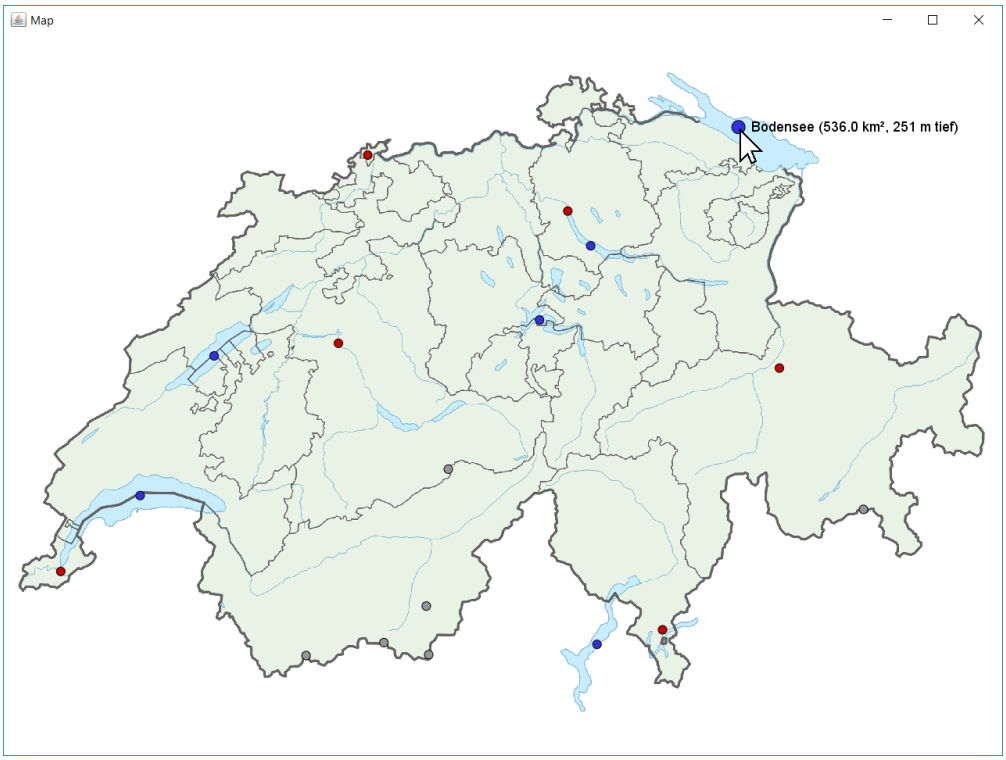
   Die Methode gibt -1 zurück, falls kein Kunde die Kriterien erfüllt. Falls mehrere Kunden die Kriterien erfüllen, dann kann eine beliebige ID dieser Kunden zurückgegeben werden.

   Ändern Sie die Methode `process` so, dass für jeden Kunden der Rechnungsbetrag in einer geeigneten Datenstruktur gespeichert wird. Die Methode `query` soll dann diese Datenstruktur verwenden um die Rückgabewerte zu berechnen. Es ist Ihnen überlassen, welche Datenstruktur Sie zur Speicherung der Informationen über die Kunden benutzen. Eine (lineare) Liste ist akzeptabel. Es ist aber wichtig, dass Ihr Programm die Inputdatei nur einmal liest und dass Ihr Programm eine beliebige Anzahl von Kundeneinträgen verarbeiten kann.

In der Datei "BillsTest.java" finden Sie einen einfachen Test, um das Format Ihres Outputs zu testen.

Aufgabe 4: Interaktive Karte

In dieser Übung verwenden Sie die `Window`-Klasse um eine interaktive Karte zu erstellen. Auf der Karte werden verschiedene *points of interest* (POI) angezeigt. Wenn der Benutzer mit der Maus auf einen solchen POI zeigt, sollen einige Informationen dazu angezeigt werden:



In der Vorlage befindet sich bereits ein Skelett für `SwissMap`, welches eine Window-Instanz erstellt und ein Hintergrundbild darin anzeigt. Ausserdem finden Sie die `PointOfInterest`-Klasse, welche die Basis-Klasse für verschiedene Arten von POIs bildet. Ihre Aufgabe ist es, drei Subklassen von `PointOfInterest` (d.h. Klassen, die von `PointOfInterest` erben) zu erstellen: `City`, `Lake` und `Mountain`. Danach sollen Sie einige Instanzen dieser Klassen erstellen und auf Ihrer Karte anzeigen.

a) Erstellen Sie eine neue Klasse `City`, welche von `PointOfInterest` erbt. Diese Klasse soll zusätzlich zu den (geerbten) Feldern von `PointOfInterest` noch Felder für Einwohnerzahl und Fläche haben. Erstellen Sie auch einen Konstruktor, welcher Werte für alle geerbten und für alle eigenen Felder von `City` entgegen nimmt. Um die geerbten Felder zu initialisieren, sollen Sie mithilfe von `super(...)` den Superkonstruktor aufrufen.

   Gehen Sie analog für `Lake` (mit Fläche und Tiefe) und `Mountain` (mit Höhe) vor.

b) Überschreiben Sie in allen drei Klassen `City`, `Lake` und `Mountain` die beiden geerbten Methoden `color()` und `description()`.

   Die erste Methode, `color()`, soll die Farbe zurückgeben, in der ein POI angezeigt werden soll. Seen sollen blau angezeigt werden, Städte rot, usw. Beachten Sie, dass der Rückgabetyp `Color` eine Klasse ist, die von der "Window.java"-Datei kommt und einfach drei RGB-Werte speichert. `Color`-Objekte können direkt an `Window.setColor()` übergeben werden.

   Die zweite Methode, `description()`, soll eine Beschreibung zurückgeben, welche angezeigt wird, wenn der Benutzer die Maus über einen POI bewegt. Die Version in `PointOfInterest` gibt einfach den Namen des Punktes zurück. Überschreiben Sie diese Methode in den drei

Subklassen, um eine spezifischere Beschreibung für jeden Typ von POI zurückzugeben (siehe Beispiel oben).

c) Vervollständigen Sie die `SwissMap`-Klasse. Erstellen Sie erst einige POIs, z.B. so:

```
PointOfInterest[] pois = new PointOfInterest[] {
        new City("Zürich",         683354, 247353, 396030, 91.88),
        new Lake("Bodensee",       744895, 277632,    536, 251),
        new Mountain("Matterhorn", 617049,  91670,   4478)};
```

In Anhang B finden Sie eine längere Liste von POIs, die Sie kopieren können. Um die Koordinaten von weiteren POIs herauszufinden, besuchen Sie am einfachsten die entsprechende Wikipedia-Seite und kopieren die "CH1903"-Koordinaten oben rechts des Artikels.

Ergänzen Sie dann die `show()`-Methode so, dass diese POIs angezeigt werden. Um die "realen" Koordinaten in GUI-Koordinaten umzuwandeln, können Sie die `toGuiX()`- und `toGuiY()`-Methoden verwenden. Die Informationen eines POIs sollen nur angezeigt werden, wenn sich der Mauszeiger in der Nähe befindet. Verwenden Sie dazu die `getMouseX()`- und `getMouseY()`-Methoden der `Window`-Klasse, welche die aktuelle Position des Mauszeigers angeben.

Anhang A: IntList Interface

```
public interface IntList {

  /** Return the integer value at position 'index'. */
  public int get(int index);

  /** Set the integer value at position 'index' to 'value'. */
  public void set(int index, int value);

  /** Returns whether the list is empty (has no values). */
  public boolean isEmpty();

  /** Returns the size of the list. */
  public int getSize();

  /** Inserts 'value' at position 0 in the list. */
  public void addFirst(int value);

  /** Appends 'value' at the end of the list. */
  public void addLast(int value);

  /**
   * Removes and returns the first value of the list.
   * Throws a NoSuchElementException if the List is empty.
   */
```

```
    public int removeFirst();

    /**
     * Removes and returns the last value of the list.
     * Throws a NoSuchElementException if the List is empty.
     */
    public int removeLast();

    /** Removes all values from the list, making the list empty. */
    public void clear();

    /** Returns a new int-array with the same contents as the list. */
    public int[] toArray();
  }
```

Anhang B: Mehr POIs

```
PointOfInterest[] pois = new PointOfInterest[] {
        new City("Zürich", 683354, 247353, 396030, 91.88),
        new City("Genf",   500532, 117325, 201810, 15.89),
        new City("Basel",  611220, 267503, 175130, 22.75),
        new City("Bern",   600670, 199655, 141660, 51.60),
        new City("Lugano", 717505,  96295,  63580, 75.80),
        new City("Chur",   759662, 190702,  37110, 28.09),

        new Lake("Bodensee",           744895, 277632, 536,    251),
        new Lake("Genfersee",          529160, 144713, 580,    310),
        new Lake("Neuenburgersee",     555829, 195103, 217.9,  152),
        new Lake("Lago Maggiore",      693884,  91043, 212.5,  372),
        new Lake("Vierwaldstättersee", 673175, 208048, 113.72, 214),
        new Lake("Zürichsee",          691603, 234802,  88.17, 136),

        new Mountain("Dufourspitze", 633220,  87321, 4634),
        new Mountain("Dom",          632330, 104856, 4545),
        new Mountain("Matterhorn",   617049,  91670, 4478),
        new Mountain("Grand Combin", 589008,  86994, 4314),
        new Mountain("Jungfrau",     640278, 154213, 4158),
        new Mountain("Piz Bernina",  789947, 139751, 4049)};
```