Grundlagen der Informatik 1
WS 2008/09

Prof. Mühlhäuser, Dr. Rößling, Melanie Hartmann, Daniel Schreiber
http://proffs.tk.informatik.tu-darmstadt.de/gdi1

Übung 11 - Lösungsvorschlag Version: 1.0 19.01.2009

---

1 Mini Quiz

Kreuzen Sie die wahren Aussagen an.

- ☐ Die Datentypen `int`, `short` und `char` werden von `Object` abgeleitet.
- ☒ Primitive Datentypen können automatisch in Objekte verwandelt werden.
- ☒ `ArrayList`, `LinkedList` und `Vector` implementieren das Interface `List`.
- ☐ In der Datenstruktur `Set` kann ein konkretes `Objekt` mehrere Male vorhanden sein.
- ☒ Mehr über Collections findet man unter `http://java.sun.com/docs/books/tutorial/collections/index.html`

2 Fragen

1. Erklären Sie in eigenen Worten, was eine Wrapper-Klasse ist. Für welche Datentypen gibt es eine Wrapper-Klasse? Nennen sie mindestens drei konkrete Beispiele.

   Lösungsvorschlag:
   Siehe T15/23: Primitive Datentypen sind keine Referenztypen und werden deswegen nicht von Object abgeleitet. Sie können Variablen vom Typ Object nicht zugewiesen werden und besitzen keine Methoden. Es kann aber sinnvoll sein, primitive Datentypen so zu behandeln, als wären sie Objekte. Jeder primitive Datentyp besitzt eine Wrapper-Klasse, so dass dieser, wenn es nötig ist, wie ein Objekt benutzt werden kann. Beispielsweise int -> Integer, byte -> Byte, double -> Double.

2. Wo liegen Unterschiede zwischen dem Java Interface `java.util.Collection` und `list` aus Scheme?

   Lösungsvorschlag:
   Siehe T15/26: Der Vergleich gestaltet sich als relativ schwierig. Collection ist ein allgemeines Interface zum Sammeln von Objekten. Die in Scheme verwendete Liste kann auch als Sammlung von verschiedensten Daten benutzt werden, aber sie ist an Nutzungsbedingungen gebunden. Die Nutzungsmöglichkeiten von Scheme-Listen ist vorgegeben und implementiert. Die tatsächlich implementierten Collections können auf unterschiedliche Art und Weise implementiert sein und zusätzliche Funktionalitäten bezüglich Nutzung, Zugriffsgeschwindigkeit und Inhaltsvorgaben bieten.

3. Erläutern Sie mögliche Implementierungen von `List` und `Set`.

   Lösungsvorschlag:
   Siehe T15/26: Objekte in einer Liste haben eine Ordnung, jedes Objekt befindet sich an einer bestimmten Position in der Liste. Diese Position ist der Index des Objektes. Ein Objekt kann mehrfach in einer Liste enthalten sein.

   Siehe T15/32: Ein Set repräsentiert eine mathematische Menge. Daher kann jedes Objekt nur einmal in einem Set enthalten sein. Die Objekte in einem Set haben keine wohldefinierte Ordnung, es gibt also auch kein „erstes" Objekt.

3 LinkedList

Bevor Sie anfangen, etwas zu implementieren, schauen Sie sich die JavaDoc-Dokumentation zur Klasse `LinkedList` an.
Listen in Java können beliebige Objekttypen speichern. Das ist praktisch, um Objekte in eine Liste hinein zu tun, aber unpraktisch, wenn man die Objekte wieder hinausholen will. Beim Herausholen muss man nämlich eine Umwandlung in den ursprünglichen Typ durchführen. Ab Java 1.5 kann man dies durch die Verwendung von „generics" (die kommen in T18) umgehen. Hier sollen Sie die Klasse `StringList`, eine Lösung für Strings ohne „generics", implementieren.

```
1  import java.util.LinkedList;
2
3  public class StringList exteds LinkedList {
4      /**
5       * appends a String to the list
6       * @param s String to add to the list
7       */
8      public void add(String s) {
9          //TODO implement method
10     }
11
12     /**
13      * Returns the element at the specified position in this list.
14      * @param index index of element to return.
15      */
16     public String get(int index) {
17         //TODO implement method
18     }
19
20     /**
21      * sort the Strings in the list alphabetically.
22      */
23     public void sortList() {
24           //TODO implement method
25     }
26 }
```

1. Implementieren Sie die Methoden `add` und `get`. Verwenden Sie dabei die Methoden der Basisklasse.

2. Implementieren Sie die Methode `sortList`, die die Liste alphabetisch aufsteigend sortiert.

Lösungsvorschlag:

```
1  import java.util.LinkedList;
2
3  public class StringList exteds LinkedList {
4     /**
5      * appends a String to the list
6      * @param s String to add to the list
7      */
8     public void add(String s) {
9        super.add(s);
10    }
11
12    /**
13     * Returns the element at the specified position in this list.
14     * @param index index of element to return.
15     */
16    public String get(int index) {
17       return (String) super.get(index);
18    }
19
20    /**
21     * sort the Strings in the list alphabetically.
22     */
23    public void sortList() {
24          java.util.Collections.sort(this);
25    }
26 }
```

4 HashMap

Bevor Sie anfangen, etwas zu implementieren, schauen Sie sich die JavaDoc Dokumentation zu `HashMap` an.
Betrachten sie die Klasse `WordVector`. Die Klasse hat drei Methoden: `parseText`, `wordFrequency` und `wordVector`. Die Klasse hat ein Feld vom Typ `HashMap`.

```
1  public class WordVector {
2     private HashMap frequencyMap = new HashMap();
3
4     public void parseText(String[] text) {
5        for (String word: text) {
6           if (frequencyMap.containsKey(word)) {
7              frequencyMap.put(word,
8                               (Integer)frequencyMap.get(word) + 1);
9           } else {
10             frequencyMap.put(word, 1);
11          }
12       }
13    }
14
15    public int wordFrequency(String word) {
16       return (Integer) frequencyMap.get(word);
17    }
18
19    public Vector wordVector(String[] index) {
```

```
20        Vector wordVector = new Vector();
21        for (String word: index) {
22           if (frequencyMap.containsKey(word) &&
23               (Integer)frequencyMap.get(word) > 0)
24              wordVector.addElement(1);
25           } else {
26              wordVector.addElement(0);
27           }
28        }
29        return wordVector;
30     }
31  }
```

1. Ergänzen Sie fehlende Kommentare für die Methoden.
2. Um primitive Datentypen in eine Liste speichern zu können, werden sie automatisch „geboxt“. Um sie weiterverwenden zu können, müssen sie wieder „unboxed“ werden. An welchen Stellen passiert dies?
3. Warum wird eine `HashMap` und keine Liste verwendet?
4. Die Klasse soll effizienter werden. Da niemand die Methode `wordFrequency` verwendet, soll diese entfernt werden. Damit kann auch auf das Zählen der Häufigkeit des Vorkommens eines Wortes verzichtet werden. Statt einer `HashMap` soll der Datentyp `Set` für das Feld `frequencyMap` verwendet werden. Ändern Sie die Implementierung von `parseText` und `wordVector` entsprechend.

Lösungsvorschlag:

```
1  public class WordVectorEfficient {
2        private Set frequencyMap = new HashSet();
3
4     /**
5     * counts the frequency for every word in a text
6     * @param text array containing the words of the text
7     */
8     public void parseText(String[] text) {
9           for (String word: text) {
10             frequencyMap.add(word);
11          }
12    }
13
14    /**
15    * transforms this text into a word vector containing 1 or 0 for every
16    * word in the index.
17    * @param index index used for computing the word vector
18    * @return the word vector for this text
19    */
20    public Vector wordVector(String[] index) {
21       Vector wordVector = new Vector();
22       for (String word: index) {
23         if (frequencyMap.contains(word)) {
24           wordVector.addElement(1);
25          } else {
```

```
26                wordVector.addElement(0);
27            }
28        }
29        return wordVector;
30    }
31 }
```

5 StringStack

Ein Stack ist eine spezielle Datenstruktur aus dem Collections Framework. Sie sollen hier eine spezielle Variante von Stack implementieren, die nur Strings speichert. Intern sollen Sie einen Vektor (`Vector`) zur Speicherung der Daten verwenden. Lesen Sie die Dokumentation der Klasse Vector und überlegen Sie dann, an welcher Position die Spitze des Stacks gespeichert werden soll.

```
1  import java.util.Vector;
2
3  public class StringStack {
4
5      private Vector theStack  = new Vector();
6
7      public void push(String s) {
8       //TODO implement
9      }
10
11     public String top() {
12      //TODO implement
13     }
14
15     public String pop() {
16      //TODO implement
17     }
18
19     public int size() {
20      //TODO implement
21     }
22
23     public boolean empty(){
24      //TODO implement
25     }
26
27 }
```

- Vervollständigen sie oben stehenden Code, sodass die Funktionalitäten eines Stack für Strings implementiert werden. Ergänzen Sie auch die Kommentare!
  - push(String s): Legt den String s auf der Spitze des Stacks ab.
  - top(): Liefert den String, der auf der Spitze des Stacks liegt. Der String bleibt auf der Spitze des Stack liegen.
  - pop(): Liefert den String der auf der Spitze des Stack liegt, dabei wird der Wert von der Spitze des Stack entfernt.
  - size(): Liefert die Anzahl der Elemente im Stack.
  - empty(): Liefert true, wenn der Stack leer ist, sonst false.

Lösungsvorschlag:

```java
/**
* Implements a stack internally using a vector. The top of the stack is stored
* at position 0 in the vector.
*/
public class StringStack {

    private Vector theStack  = new Vector();

    /**
     * Adds a new element on top of the stack
     * @param s String to push on top of stack.
     */
    public void push(String s) {
      theStack.add(0,s);
    }

    /**
     * Return the String on top of the stack
     * @return the String on top of the stack
     */
    public String top() {
     return (String)theStack.get(0);
    }

    /**
     * Return the element on top of the stack and remove it from the stack
     * @return the element on top of the stack
     */
    public String pop() {
       return (String)theStack.remove(0);
    }

    /**
     * @return whether this stack is empty
     */
    public boolean empty(){
       return theStack.isEmpty();
    }
}
```

```java
/**
* Implements a stack internally using a vector. The top of the stack is stored
* at the last position in the vector.
*/
public class StringStack {

    private Vector theStack  = new Vector();

    /**
     * Adds a new element on top of the stack
     * @param s String to push on top of stack.
     */
    public void push(String s) {
```

```
14        theStack.add(s);
15    }
16
17    /**
18     * Return the String on top of the stack
19     * @return the String on top of the stack
20     */
21    public String top() {
22     return (String)theStack.get(theStack.size() - 1);
23    }
24
25    /**
26     * Return the element on top of the stack and remove it from the stack
27     * @return the element on top of the stack
28     */
29    public String pop() {
30       return (String)theStack.remove(theStack.size() - 1);
31    }
32
33    /**
34     * @return whether this stack is empty
35     */
36    public boolean empty(){
37       return theStack.isEmpty();
38    }
39 }
```

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren.

Abgabe: Bis spätestens Fr, 30.01.2009, 16:00

6 Bar-Chart (3P)

Da es bei dieser Aufgabe um grafische Ausgabe geht, für die JUnit Tests nur umständlich zu realisieren sind, sollen Sie auf die Implementierung von eigenen Tests verzichten!
Die Vorlage zur Hausübung enthält die Klasse `BarChart`. Die Klasse erbt von `GraphicsProgram`. Die `main` Methode enthält einen Beispielaufruf zur Erzeugung eines Balkendiagramms. Zunächst wird ein `BarChart` Objekt erzeugt und die grafische Ausgabe gestartet. Anschließend wird ein Diagramm durch Aufrufen der Methode `displayDiagram(Map, double)` angezeigt. Die Daten werden als `Map` übergeben. Die Schlüssel in der `Map` werden als Rubriken auf der X-Achse angezeigt, die Werte als Balkenhöhe. Es dürfen nur Werte vom Typ `Double` in der Map enthalten sein! Der numerische Parameter gibt den größten y-Wert an.
Das Problem, das in dieser Aufgabe gelöst werden soll, ist die Reihenfolge der Balken festzulegen. Die Reihenfolge der Schlüssel in einer `HashMap` ist nämlich zufällig.

1. Ändern Sie die Klasse so ab, dass die Reihenfolge der Rubriken festgelegt werden kann. Eine Möglichkeit ist es, einen dritten Parameter für die Reihenfolge an `displayDiagram` zu übergeben.

2. Ändern Sie die Methode `run`, so dass die Reihenfolge der Rubriken bei der Ausgabe berücksichtigt wird.

Lösungsvorschlag:

```
1
2  import java.util.HashMap;
3  import java.util.Map;
4
5  import acm.graphics.GLabel;
6  import acm.graphics.GRect;
7  import acm.program.GraphicsProgram;
8
9  public class BarChart extends GraphicsProgram {
10    private static final long serialVersionUID = 1L;
11    private static int WIDTH = 600;
12    private static int HEIGHT = 400;
13    private static int GAP = 20;
14
15
16    /**
17     * Display a bar chart of data. The parameter ymax specifies the
        scaling of the yaxis.
18     *
19     * @param data the data to display
20     * @param ymax the maximum of the y-axis
21     */
22    public void displayDiagram(Map<String, Double> data, double ymax) {
23       //clean canvas
24       removeAll();
25       // compute bar width
26       int barWidth = (WIDTH - GAP * (data.size() - 1)) / data.size();
27       // compute bar unit height
28       double barUnitHeight = HEIGHT / ymax;
29       // x of lower left corner of next bar
30       int x = 0;
31
32       // draw a rectangle with label for every entry
33       for (Map.Entry<String,Double> o: data.entrySet()) {
34 //        Map.Entry e = (Map.Entry) o;
35          // bar height corresponds to value of the entry
36          Double v = (Double) o.getValue();
37          GRect bar = new GRect(barWidth, barUnitHeight * v);
38          add(bar, x, HEIGHT + 1 - barUnitHeight * v);
39          // label text is the key of the entry
40          GLabel lbl = new GLabel((String) o.getKey());
41          //center label in bar
42          add(lbl, x + (barWidth - lbl.getWidth()) / 2, HEIGHT);
43          x = x + barWidth + GAP;
44       }
45    }
46
```

```
47      /**
48       * Opens a window with an example bar chart.
49       * You do not need to change this method!
50       * @param args command line arguments are not used in this exercise
51       */
52      public static void main(String[] args) {
53          BarChart b = new BarChart();
54          b.start();
55
56          Map<String, Double> data = new HashMap<String, Double>();
57          data.put("0 - 4", 3.0);
58          data.put("5 - 9", 7.0);
59          data.put("10 - 14", 1.0);
60
61          b.displayDiagram(data,10);
62      }
63  }
```

6.1 Temperatur Sensor (7P)

In einem Gebäube werden Temperatursensoren verteilt, die die gemessene Temperatur, zusammen mit dem Ort an dem Sie aufgebaut, sind in regelmäßigen Zeitabständen an eine Zentrale funken. In der Zentrale befindet sich ein Display, das für jeden Sensor die Durchschnittstemperatur der letzten 30 Messwerte anzeigt. Das System ist so konzipiert, dass man einfach weitere Sensoren aufstellen und starten kann, ohne an der Anzeige-Software etwas zu ändern.
Von der Vorlesungshomepage erhalten Sie die Klassen `TemperatureTicker` und `InteractiveChart`. `TemperatureTicker` simuliert die im Gebäude verteilten Temperatursensoren. Die Implementierung von `TemperatureTicker` ist für die Lösung der Aufgabe nicht wichtig. Die Klasse sorgt dafür, dass die Methode `newTemperature` von `InteractiveChart` aufgerufen wird. Als Parameter wird bei jedem Aufruf die Zeit der Messung (in Millisekunden seit Mitternacht, 1. Januar 1970), der Ort der Messung und die aktuelle Temperatur übergeben. Ihre Aufgabe ist es, die Anzeige-Software zu implementieren. Erweitern Sie dazu die Klasse `InteractiveChart`, die von `BarChart` erbt.
Implementieren Sie die Funktion `newTemperature`, die bei jedem eintreffenden Messwert ein Balkendiagramm ausgibt. Die aktuelle Zeit (in Millisekunden seit Mitternacht, 1. Januar 1970) erhalten Sie durch Aufruf von `System.currentTimeMillis()`.
Da sich die Anzeige des Balkendiagramms nur schwierig in Testfällen überprüfen lässt, können Sie davon ausgehen, dass die Anzeigefunktion an sich funktioniert. Daher müssen Sie nur die Parameter, die an diese Funktion übergeben werden, überprüfen.
Die Tutoren werden die Funktionalität ihrer Software nach dem Schema unten bewerten, aber für unverständlichen oder schlecht kommentierten Code Punkte abziehen. Die Basisfunktionalität wird durch die in der Klasse `TestInteractiveChart` angegebenen Testfälle überprüft. Sie dürfen gerne weitere Testfälle erstellen.

- 1 Punkt: Es wird ein beliebiges Balkendiagramm angezeigt
- 2 Punkte: Es wird zu jedem Ort, von dem jemals Temperaturen empfangen wurden, ein Balken angezeigt.
- 3 Punkte: Die Balken sind alphabetisch sortiert.
- 5 Punkte: Es wird zu jedem Ort die an diesem Ort zuletzt gemessene Temperatur angezeigt.

- 6 Punkte: Es wird zu jedem Ort der Durchschnitt aller an diesem Ort gemessenen Temperaturen angezeigt.
- 7 Punkte: Es wird zu jedem Ort der Durchschnitt der letzten 30 an diesem Ort gemessenen Temperaturen angezeigt, bzw. der Durchschnitt aller an diesem Ort gemessenen Temperaturen, wenn noch keine 30 Messwerte von diesem Ort vorliegen.

Lösungsvorschlag:

```
1  import java.util.Collections;
2  import java.util.HashMap;
3  import java.util.LinkedList;
4  import java.util.Map;
5
6  public class InteractiveChart extends BarChart
7        implements TemperatureTicker.TemperatureClient,
           InteractiveChartTestInterface {
8
9     /*
10     * Stores a list with up to 30 Measurements for each location.
11     * This list is used to compute the average temperature for this
         location.
12     */
13    private HashMap tempHistoryPerLocation = new HashMap();
14
15    /*
16     * Alphabetically sorted list of all locations. This variable is
         needed,
17     * as the keys in the tempHistoryPerLocation map are not ordered.
         Whenever
18     * a Measurement from a new location is received this list needs to be
          re-sorted.
19     */
20    private LinkedList allLocations = new LinkedList();
21
22    /*
23     * stored for testing purposes
24     */
25    private Map lastDiagramData;
26
27    /**
28     * Record for storing measurements from temperature sensors
29     */
30    class Measurement {
31       Measurement(long t, String l, double temp) {
32          time = t;
33          location = l;
34          temperature = temp;
35       }
36       public long time;
37       public String location;
38       public double temperature;
39    }
40
41    /**
42     * @return the locations used for the last displayed diagram
```

```
43      */
44     public LinkedList getLastDiagramLocations() {
45        return allLocations;
46     }
47
48     /**
49      * @return the data used for the last displayed diagram
50      */
51     public Map getLastDiagramData() {
52        return lastDiagramData;
53     }
54
55     /**
56      * Open a new window and start the TemperatureTicker.
57      * Do not change this method!
58      * @param args command line parameters are not used in this exercise
59      */
60     public static void main(String[] args) {
61        InteractiveChart chart = new InteractiveChart();
62        chart.start();
63        //start the temperature ticker
64        TemperatureTicker.initSensor(chart);
65     }
66
67     /**
68      * Called whenever a new temperature is measured by a sensor.
69      * @param timestamp The time when the measurement was taken, given in
          microseconds since 1.1.1970
70      * @param location The location where the measurement was taken
71      * @param temperature The measured temperature
72      */
73     public void newTemperature(long timestamp, String location, double
        temperature) {
74        //add location if not yet there
75        if (!allLocations.contains(location)) {
76           allLocations.add(location);
77           //history for a new location is empty
78           tempHistoryPerLocation.put(location, new LinkedList());
79        }
80        //sort locations alphabetically
81        Collections.sort(allLocations);
82
83        //add new temperature for this location
84        LinkedList locationHistory = (LinkedList) tempHistoryPerLocation.
            get(location);
85        locationHistory.add(new Measurement(timestamp, location,
            temperature));
86        // store 30 measurements max, if the history contains more than 30,
             remove
87        // the oldest data point
88        if (locationHistory.size() > 30) {
89           locationHistory.remove(0);
90        }
91
92        //create data map for displaying the diagram
93        Map data = new HashMap();
94        for (Object s: allLocations) {
95           locationHistory = (LinkedList) tempHistoryPerLocation.get(s);
```

```
96              double sum = 0;
97              int count = 0;
98              //average over all values in the history
99              for (Object q: locationHistory) {
100                   sum += ((Measurement) q).temperature;
101                   count++;
102             }
103             //display the location as category and the temperature average
                  as value
104             //in the diagram.
105             data.put(s, sum / count);
106         }
107
108         //draw bar chart, use alphabetic list of locations for sorting
              categories
109         displayDiagram(data, 100, allLocations);
110
111         //store the data for testing purposes
112         lastDiagramData = data;
113     }
114 }
```