Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Übung 11 Version: 1.0 18. 01. 2010

---

1 Mini Quiz

Bitte 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.
2. Wo liegen die Unterschiede zwischen dem Java Interface `java.util.Collection` und `list` aus Scheme?
3. Erläutern Sie mögliche Implementierungen von `List` und `Set`.

3 LinkedList

Bevor Sie anfangen, etwas zu implementieren, schauen Sie sich bitte die JavaDoc-Dokumentation zur Klasse `java.util.LinkedList` und ihrer Obertypen an.
Listen in Java können beliebige Objekttypen speichern. Das ist praktisch, um Objekte in eine Liste einzufügen, aber unpraktisch, wenn man auf die Objekte zugreifen will. Beim Zugriff muss nämlich eine Umwandlung in den ursprünglichen Typ durchgeführt werden. Ab Java 1.5 kann man dies durch die Verwendung von „Generics“ (→ 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 extends 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 bitte implementieren
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 bitte implementieren
18    }
19
20    /**
21     * sort the Strings in the list alphabetically.
22     */
23    public void sortList() {
24      //TODO bitte implementieren
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.

   Hinweis: Beachten sie dazu die Dokumentation der Collections!

4 HashMap

Bevor Sie anfangen, etwas zu implementieren, schauen Sie sich bitte 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  import java.util.HashMap;
2  import java.util.Vector;
3
4  public class WordVector {
5    private HashMap frequencyMap = new HashMap();
6
7    public void parseText(String[] text) {
8      for (String word: text) {
9        if (frequencyMap.containsKey(word)) {
10         frequencyMap.put(word,
11                          (Integer)frequencyMap.get(word) + 1);
12       } else {
13         frequencyMap.put(word, 1);
14       }
15     }
16   }
17
18   public int wordFrequency(String word) {
19     return (Integer) frequencyMap.get(word);
20   }
21
22   public Vector wordVector(String[] index) {
23     Vector wordVector = new Vector(frequencyMap.size());
24     for (String word: index) {
25       if (frequencyMap.containsKey(word)
26           && (Integer)frequencyMap.get(word) > 0) {
27         wordVector.addElement(1);
```

```
28          } else {
29            wordVector.addElement(0);
30          }
31        }
32        return wordVector;
33      }
34  }
```

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.

5 StringStack

Ein Stack ist eine Datenstruktur aus dem Collections Framework. Sie sollen hier eine spezielle Variante von Stack implementieren, die nur Strings speichert. Intern sollen Sie zur Speicherung der Daten einen Vektor (`java.util.Vector`) 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 bitte implementieren
9    }
10
11   public String top() {
12     //TODO bitte implementieren
13   }
14
15   public String pop() {
16     //TODO bitte implementieren
17   }
18
19   public int size() {
20     //TODO bitte implementieren
21   }
22
23   public boolean empty(){
24     //TODO bitte implementieren
25   }
26
27 }
```

Vervollständigen sie den oben stehenden Code, so dass die Funktionalitäten eines Stack für Strings implementiert werden. Ergänzen Sie auch die Kommentare!

void push(String s) legt den String s auf der Spitze des Stacks ab.

String top() liefert den String, der auf der Spitze des Stacks liegt. Der String bleibt auf der Spitze des Stack liegen.

String pop() liefert den String der auf der Spitze des Stack liegt. Dabei wird der Wert von der Spitze des Stack entfernt.

int size() liefert die Anzahl der Elemente im Stack.

boolean empty() liefert **true**, wenn der Stack leer ist, sonst **false**.

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. Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Portal auch für sie bewertet werden kann.

Abgabedatum: Freitag, 29. 01. 2010, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß den Vorgaben zu kommentieren (Scheme: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Java: JavaDoc). Zerlegen Sie Ihren Code sinnvoll und versuchen Sie, wo es möglich ist, bestehende Funktionen wiederzuverwenden. Wählen Sie sinnvolle Namen für Hilfsfunktionen und Parameter.

6 Börsenticker (12 Punkte)

In dieser Übung sollen Aktienkurse von fünf verschiedenen Unternehmen mit Hilfe des `acm.graphics` Package[1] [2] visualisiert werden. Wie üblich können Sie hierfür wieder eine Vorlage aus dem Portal herunterladen.

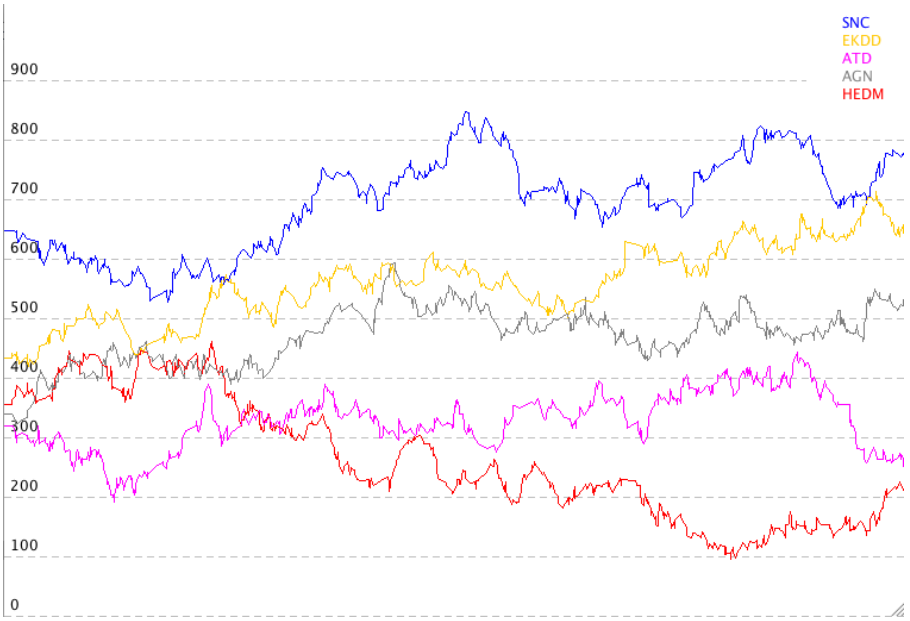
6.1 Projekt anlegen

Erstellen Sie zunächst ein neues Java-Projekt und importieren Sie das Template aus dem Portal. Binden Sie die `acm.jar` sowie die `JUnit`-Bibliotheken in Ihr Projekt ein, wie im Portal beschrieben. Benennen Sie anschließend die Klasse `StockTickerTemplate` in `StockTicker` um.

6.2 Diagramm vorbereiten (5 Punkte)

Wie in der Darstellung zu sehen, soll das Diagramm eine durchgezogene Y-Achse mit einer geeigneten Unterteilung am linken Rand enthalten. Die X-Achse soll durch Zwischenmarkierungen im Abstand von 50 Pixeln in einem helleren Grau als unterbrochene Linien bis zum rechten Bildschirmrand gezeichnet werden. Außerdem soll es in der oberen rechten Ecke eine Legende für die farbkodierten Tickersymbole geben.
Beginnen Sie damit, in `init()` die Y-Achse als durchgezogene schwarze Linie von der oberen linken zur unteren linken Ecke zu zeichnen. Für diesen Zweck steht die Klasse `acm.graphics.GLine` sowie die Methode `getHeight()` zur Verfügung (0.5 Punkte).

[1] `http://jtf.acm.org/javadoc/student/index.html`
[2] `http://jtf.acm.org/rationale/graphics-package.html`



Als nächstes sollen Sie mehrere horizontale Markierungen im Abstand von 50 Pixeln zeichnen. Diese sollen aus vielen kleinen Liniensegmenten bestehen, um den Eindruck einer unterbrochenen Linie zu erzeugen. Wählen Sie als Farbe ein helles Grau.
Die X-Achse selbst soll am unteren Rand der Zeichenfläche platziert werden und den Wert 0 repräsentieren. Der obere Wert der Zeichenfläche wird durch die Klassenkonstante `StockTicker.MAX_WORTH` bestimmt; entsprechend ergeben sich die Werte für die dazwischen liegenden Markierungen. Versehen Sie die Markierungen ebenfalls etwas eingerückt und hochgestellt am linken Rand mit dem von ihnen repräsentierten Wert als `acm.graphics.GLabel` (2 Punkte).
Hinweis: Die aktuelle Version des `acm.jar` liefert anscheinend bei `getHeight()` nicht die korrekte Höhe der Zeichenfläche, sondern einen etwas größeren Wert. Ignorieren Sie diesen Fehler und nehmen Sie an, dass `getHeight()` korrekt funktioniert. Nachdem alle Kurse gezeichnet wurden, können Sie das Fenster manuell vergrößern, um die unterste X-Achse anzeigen zu lassen.
Zuletzt müssen Sie noch die Legende der Farbkodierung zeichnen lassen. Beginnen Sie damit, ein ausreichend großes, weiß ausgefülltes Rechteck (`acm.graphics.GRect`) in der rechten oberen Ecke zu platzieren, um die oberen X-Achsen zu überdecken. Iterieren Sie nun über `StockTicker.SYMBOLS` und zeichnen Sie ein `acm.graphics.GLabel` in der Farbe, wie sie in in der `colors`-Map für das Tickersymbol hinterlegt ist (2.5 Punkte).

6.3 Graphen zeichnen (6 Punkte)

Sobald ein Wert für ein Tickersymbol eines Unternehmens ermittelt wurde, wird in der Vorlage die Methode **public void** updateTicker(String symbol, **double** value) aufgerufen, damit Sie den entsprechenden Graphen aktualisieren können. Die Aktualisierungen der Kurse unterliegen keinem festen Intervall, sie werden quasi geliefert sobald neue Daten verfügbar sind.
Die Graphen in unserem Ticker bestehen aus vielen kleinen Linien, welche einfach für jede Aktualisierung von der alten zur neuen Koordinate in der entsprechenden Farbe des Tickersymbols gezeichnet werden. Daher müssen Sie Vorkehrungen treffen, die Koordinaten des Ende des letzten Liniensegmentes des Tickersymbols zwischenzuspeichern.
Außerdem müssen Sie sich Gedanken machen, welcher X-Wert einem Aufruf zuzuordnen ist. Da das Diagramm mit der Zeit voranschreiten soll, müssen Sie für jeden Aufruf `System.currentTimeMillis()` mit `startedAt` vergleichen, um zu schauen, wieviel Millisekunden seit Beginn der Aufzeichnungen verstrichen sind. Die maximale Zeitdifferenz und damit der Zeitpunkt des letzten Aufrufs von `updateTicker` ist in der statischen Klassenkonstante MAX_DURATION in Millisekunden vermerkt.

1. Legen Sie ein neues Attribut `Map<String, acm.graphics.GPoint>` als `oldPos` an, um die Endkoordinaten der letzten Liniensegmente zwischenzuspeichern. Initialisieren Sie es mit der konkreten Implementierung einer `HashMap` im Konstruktor. Alternativ können Sie auch zwei `Map<String, Double>` Container als `oldX` und `oldY` verwenden (1 Punkte).

   Hinweis: Die Notation Map<String, GPoint> bedeutet, dass der verwendete "Schlüssel" vom Typ String und der Wert der Map vom Typ GPoint ist. Siehe auch Foliensatz T18 "Generics". Keine Sorge, Eclipse hilft hier beim korrekten Anlegen der Instanz, wenn Sie die Code Completion nutzen.

   Hinweis: Punkte auf der Zeichenfläche eines `acm.program.GraphicsProgram` Objektes werden per `double` spezifiziert[3].

2. Implementieren Sie nun **public void** updateTicker(String symbol, **double** value). Zuerst sollten Sie die Start- und Endkoordinaten des zu zeichnenden Liniensegmentes errechnen. Die Startkoordinaten sind die Endkoordinaten des letzten Durchlaufs mit diesem Tickersymbol; speichern Sie diese für den nächsten Durchlauf immer in `oldPos`. Für den ersten Durchlauf nehmen Sie einfach an, dass der Wert der gleiche gewesen wäre. Die neue X-Koordinate müssen Sie aus der Differenz der aktuellen Zeit per `System.currentTimeMillis()` mit der Startzeit der Simulation in `startedAt` und der Breite der Zeichenfläche per `getWidth()` sowie der Dauer der Simulation in `StockTicker.MAX_DURATION` errechnen. Die neue Y-Koordinate ist der Wert der Aktie, normiert auf `MAX_VALUE` und vom unteren Rand aus abgetragen.

   Zeichnen Sie das Liniensegment in der Farbe des Tickersymbols und speichern Sie die Koordinaten in `oldPos` (5 Punkte).

Wenn Sie nun als letzte Anweisung in `init()` den Aufruf von `startTicking()` einsetzen, sollten Sie beim Ausführen sehen können, wie die Graphen der Tickersymbole über `maxDuration / 1000` Sekunden gezeichnet werden.
Hinweis: Insgesamt erhalten Sie einen Punkt auf die Dokumentation Ihrer Methoden; dies betrifft insbesondere von Ihnen geschriebene Hilfsmethoden (1 Punkt).

[3] `http://jtf.acm.org/rationale/graphics-package.html#Coordinates`