Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2011/

Java-Übung Blatt 6 (Zirkuläre Klassen)

2011-06-06

Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Exportieren Sie das Projekt als ZIP-Datei mit Namen *accountname-projectname*.zip. Beispiel: Peter Lustig (lustigp) gibt ex02_1 ab als Datei lustigp-ex02_1.zip
- ZIP-Export in Eclipse: Rechtsklick auf die Wurzel des Projects, "Export...", "General/Archive File..."
- Mailen Sie die ZIP-Files an Ihren Tutor.

Abgabe: Montag, 20. Juni 2011, um 9.00 Uhr.

Aufgabe 1 (Stammbaum, 4 Punkte)
Projekt: `ex06_1`. Package: `genealogy`.
Wir betreiben Ahnenforschung. Für ein System zur Stammbaumverwaltung sollen Sie eine Klasse `Person` erstellen, die Personen im Stammbaum repräsentiert. Jede Person hat ein Geschlecht und soll sowohl seine (biologischen) Eltern als auch seine (biologischen) Kinder kennen. Kinder sollen als verkette Liste von Personen nach dem Schema aus der Vorlesung repräsentiert werden (`IPersonList`, `MTPersonList`, `ConsPersonList`). Der Wert `null` dient dazu unbekannte (Eltern) oder nicht vorhandene (Ehepartner) Personen zu repräsentieren.
Weiterhin soll jede Person heiraten können und dabei nicht vergessen, wer geheiratet wurde. Stellen Sie dazu eine öffentliche Methode `public void marry(Person spouse)` zur Verfügung. Außerdem soll soll sich eine Person scheiden lassen können: `public void divorce()`. Wenn eine Person heiratet lässt Sie sich implizit zunächst vom (eventuell vorhandenen) bisherigen Ehepartner scheiden.
Eine Person soll mit dem öffentlichen Konstruktor `public Person(boolean isMale, Person mother, Person father)` instanziert werden. Da wir nur biologische Kinder betrachten, ist dies die einzige Möglichkeit, dass der Liste der Kinder einer Person ein Kind hinzugefügt wird. Achten Sie also auf eingeschränkte Sichtbarkeit der Methode, die eine Person zu der Liste der Kinder hinzufügt. Beachten Sie die zirkulären Referenzen zwischen Personen bei allen Operationen auf Personen. Beispielsweise soll die Scheidung einer Person vom Ehepartner dazu führen, dass anschließend beide Personen keinen Ehepartner mehr haben.

(a) Implementieren Sie die Klasse `Person` nach den obigen Vorgaben. Achten Sie darauf keine `public` Felder zu verwenden. Das Geschlecht einer Person soll mit der öffentlichen Methode (`boolean isMale()` abgefragt werden können. Um die Mutter, den Vater und Eheparter abzufragen, sollen die öffentlichen Methoden `getMother`, `getFather` und `getSpouse` implementiert werden.

(b) Implementieren Sie eine öffentliche Methode `int maleChildren()`, die die Anzahl der männlichen Kinder einer Person liefert. Führen zu diesem Zweck eine abstrakte Klasse `APersonList` ein, die die Berechnung der Anzahl der männlichen Personen der Liste übernimmt und erweitern Sie das Interface `IPersonList` um Methoden für den Listendurchlauf.

Testen Sie alle Methoden und Konstruktoren Ihrer `Person`-Implementierung. Überprüfen Sie dabei insbesondere, ob die zirkulären Referenzen korrekt verändert wurden. Beachten Sie, dass `null` und gültige `Person`-Instanzen in der Regel jeweils einen eigenen interessanten Testfall darstellen, den Sie mit Ihren Tests abdecken sollen.

Aufgabe 2 (Zustandsbehaftete Klasse, 4 Punkte)
Projekt: `ex06_2`. Package: `shape`.
Modellieren sie eine Klasse `Rect`, die ein Rechteck repräsentiert. Die Klasse hat jeweils ein Feld für die x-Koordinate und die y-Koordinate der oberen linken Ecke sowie zwei weitere Felder für die Höhe und Breite. Die vier Felder sind als `double`-Werte ausgeführt. Die Klasse soll die öffentlichen Methoden `void moveTo(double x, double y` (verschiebt das Rechteck zu der angegebenen Position), `void setSize(double width, double height)` (setzt die Größe des Rechtecks) und `Rect biggestSubRect(double aspectRatio)` (liefert das größte enthaltene Rechteck mit dem angegebenen Seitenverhältnis) anbieten. Weiterhin sollen get-Methoden (`public`) für die x-Koordinate und die y-Koordinate der oberen linken Ecke sowie für die Höhe und Breite implementiert werden.
Das Seitenverhältnis beschreibt die Breite geteilt durch die Höhe des Rechtecks. Ein Aufruf der Methode `biggestSubRect` soll das größtmögliche vollständig im Rechteck enthaltene Rechteck zurückliefern, welches das angegebene Seitenverhältnis hat. Das zurückgelieferte Rechteck hat also die Breite des Originalrechtecks, wenn das neue Seitenverhältnis größer ist als das alte Seitenverhältnis. Ist das neue Seitenverhältnis kleiner als das alte Seitenverhältnis, dann entspricht die Höhe des neuen Rechtecks der Höhe des Originalrechtecks. Die Methode soll immer ein neues Rechteck instanzieren.
Wir nehmen an, dass die Methode `biggestSubRect` häufig von Ihrer Anwendung auf demselben Rechteck aufgerufen wird (insbesondere deutlich häufiger als die Methode `setSize`) und dass diese Aufrufe auf einem zeitkritischen Ausführungspfad der Anwendung liegen. Wir nehmen weiterhin an, dass die Operation die Plattform, auf der ihr Code vorwiegend ausgeführt werden soll keine effiziente (schnelle) Hardware-Implementierung für die Division zweier `double`-Werte besitzt. Die Klasse `Rect` soll daher das aktuelle Seitenverhältnis des Rechtecks als (redundantes) Feld abspeichern, um unnötige Divisionen zu vermeiden.
Testen Sie Ihren Code mit Hilfe sinnvoller Unittests. Decken Sie beim Testen der Methode `Rect getBiggestSubRect(double aspectRatio)` mindenstens 3 Fälle ab.

Aufgabe 3 (Queue, 4 Punkte)
Projekt: `ex06_3`. Package: `queue`.
Wir bauen eine Warteschlange für Strings, bei der man Elemente am Anfang und am Ende hinzufügen und entfernen kann. Die Klasse `DoubleEndedQueue` soll folgende Operationen anbieten:

- `void pushFirst(String s)` fügt `s` als vorderstes Element der Schlange hinzu.
- `void pushLast(String s)` fügt `s` als hinterstes Element der Schlange hinzu.
- `String popFirst()` entfernt das vorderste Element der Schlange und gibt es zurück. Wenn die Schlange leer ist, gibt die Methode `null` zurück.
- `String popLast()` entfernt das hinterste Element der Schlange und gibt es zurück. Wenn die Schlange leer ist, gibt die Methode `null` zurück.
- `boolean isEmpty()` ist die Queue leer?

Überlegen Sie sich selbst, wie `DoubleEndedQueue` intern aufgebaut sein soll (Sie dürfen aber keine Klassen aus `java.util` verwenden). Die Klasse muss jede Operation in konstanter Zeit durchführen können (das bedeutet insbesondere auch, sie darf für keine Operation durch die Liste aller Elemente ganz durchlaufen!). Tipp: verkettete Liste mit Referenzen auf Vorgänger und Nachfolger.
Testen Sie mit mindestens acht Fällen unterschiedliche Folgen aus Einfügen und Entfernen an verschiedenen Enden. Wenn Ihr Test keine Fälle von vergessenen Vorgänger- und Nachfolger-Updates zu Tage führt, haben Sie wahrscheinlich noch nicht gründlich genug getestet.

Nachholaufgaben für Thema 4: TwoDeeDoo-API

Mit dieser Aufgabe können Sie den Stoff von Thema 4 (dem Thema von Blatt 4) wiederholen und nochmal Punkte für Thema 4 erwerben.

Aufgabe 4 (Animationsstrategie, 4 Punkte)
Projekt: `ex04_4` Package: `anim`
In dieser Aufgabe animieren wir die Bewegung eines Käfers über den Boden, wobei die Zeit-Weg-Kurve austauschbar ist.

(a) Bauen Sie eine `Beetle`-Klasse, die das Interface `IMovable` implementiert. Der Käfer soll wissen, an welchen Koordinaten er steht. Man soll seine Koordinaten ändern können. Er soll sich an seinen gegenwärtigen Koordinaten auf ein `ICanvas` malen können.

```
1 public interface IMovable {
2   void moveTo(int x, int y);
3 }
```

Bauen Sie eine Welt `AnimWorld`, die einen Käfer besitzt und ihn sich jedesmal zeichnen lässt.

(b) Das "Bewegen eines `IMovable`-Objektes von einem Startpunkt zu einem Endpunkt innerhalb einer bestimmten Anzahl von Simulationsticks" soll von der Klasse `Animation` repräsentiert werden. Sie hat eine `onTick()`-Methode, in der sie ihrem `IMovable` neue Koordinaten gibt. Wenn die Animation vorbei ist, d.h. die vorgegebene Anzahl Simulationsticks abgelaufen sind, soll sie in `onTick()` gar nichts mehr tun. Man soll mit Methode `isFinished()` fragen können, ob die Animation schon vorbei ist.

Die Art, wie `Animation` die Koordinaten zwischen Start- und Endpunkt interpoliert, soll von einer Zeit-Weg-Kurve `ICurve` vorgegeben werden, die man dem `Animation`-Konstruktor mitgibt.

Die Kurve bildet eine normierte Zeit t von 0 (=Startzeit) bis 1 (=Endzeit) auf einen Punkt auf der aufs Intervall $[0, 1]$ normierten Strecke s von Start nach Ziel ab, wobei 0 der Start, 1 das Ziel und z.B. 0.5 die Mitte zwischen Start und Ziel ist.

```
1   public interface ICurve {
2     /**
3     * Maps a time (0.0–1.0) to a position (0.0–1.0)
4     * @param time from 0.0 to 1.0
5     * @return position on the path, from 0.0 to 1.0
6     */
7     double valueAt(double time);
8   }
```

Fangen Sie zunächst mit einer linearen Kurve `LinearCurve` an, bei der $s = t$ ist. Ihre Welt soll die Animation in jedem Tick aufrufen und ihr so Gelegenheit geben, den Käfer neu zu positionieren. Lassen Sie einen Käfer langsam irgendwo entlanglaufen.

Test: Testen Sie, dass eine kurze `Animation` mit `LinearCurve` einem `Beetle` nach jedem Tick die richtigen Koordinaten gegeben hat.

(c) Bauen Sie noch zwei andere `ICurve`-Implementationen. Irgendetwas Stetiges mit $s(0) = 0$ und $s(1) = 1$. Empfehlungen: $s = -2t^3 + 3t^2$ und $s = -4t^3 + 7t^2 - 2t$. Halten Sie den Käfer auf Trab! Wenn die Animation vorbei ist, soll sie durch eine neue Animation vom gegenwärtigen Standort zu einem zufälligen neuen Ziel ersetzt werden. Die neue Animation soll jeweils mit einer zufällig aus den drei Kurven gewählten Kurve ausgestattet sein.
Lassen Sie so den Käfer auf verschiedene Arten zickzack laufen.
Test: nur ausprobieren.