Programmieren in Java

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

Java-Übung Blatt 4 (Rekursive Klassen)

2011-05-23

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, 30. Mai 2011, um 9.00 Uhr.

Aufgabe 1 (Bäume, 4 Punkte)
Projekt: `ex04_1`. Package: `trees`.
In dieser Aufgabe sollen Sie mit dem TwoDeeDoo-Framework einen Binärbaum darstellen. Das Framework können Sie sich auf der Vorlesungsseite herunterladen. Wie dort beschrieben, können Sie entweder das JAR in Ihr Projekt einbinden oder mit dem bereitgestellten Leerprojekt beginnen.

Erstellen Sie eine Klasse `TreeWorld` die das `IWorld`-Interface implementiert. Ihre "World" soll eine Größe von "`640*480`" Pixeln haben. Diese Klasse soll maximal den Wurzelknoten des darzustellenden Baumes kennen.

Ein Baum (`ITree`) kann sein:

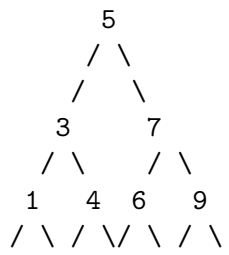
- ein Blatt `Leaf` (ohne Daten drin)
- ein innerer Knoten `InnerNode`, in dem ein `int`-Wert gespeichert ist und an dem ein rechter und ein linker Teilbaum (jeweils `ITree`) hängen.

Bäume können ihre Höhe berechnen (Blätter haben Höhe 0; Innere Knoten sind um 1 höher als ihr höheres Kind) und sich auf ein `ICanvas` zeichnen.

```
1  public interface ITree {
2      /**
3       * @return the height of this tree.
4       */
5      public int height();
6
7      /**
8       * Draws this tree within the specified rectangular area.
9       * @param x the topLeft corner's x coordinate
10      * @param y the topLeft corner's y coordinate
11      * @param width the
12      * @param height the height of the area to draw in
13      * @param canvas the canvas to draw this node's subtree in
14      */
15     public void draw(int x, int y, int width, int height, ICanvas canvas);
16 }
```

(a) Schreiben Sie Klassen für Blätter und für Innere Knoten. Testen Sie für beide die Höhenberechnung.

(b) Implementieren Sie die `draw`-Methoden. Beim Aufruf bekommt der Baum ein Rechteck übergeben (durch linke obere Ecke, Breite und Höhe). In diesem Rechteck soll der Baum sich so zeichnen, dass seine Wurzel in der Mitte der oberen Kante zu sehen ist. Falls er Kinder hat, soll er ihnen Teilrechtecke zuweisen, in denen sie sich zeichnen sollen, und er soll Verbindungslinien zu den Wurzeln der Kinder zeichnen.
Testen Sie die Darstellung ausnahmsweise durch Ausprobieren mit einer Main-Klasse, die den Baum



zeichnet.

Aufgabe 2 (Bauernhof, 4 Punkte)
Projekt: `ex04_1`. Package: `farm`.
Für diese Aufgabe setzt Ihr Projekt wieder auf dem TwoDeeDoo-Framework auf. Mit Hilfe des `IWorld`-Interfaces aus dem Framework sollen Sie ein kleines Spiel schreiben, das den Kohlanbau auf einem Acker simuliert.

Der simulierte Acker hat eine Bildschirmgröße von "`640*480`" Pixeln. Der Acker teilt sich in ein Raster von "`40*40`" großen Zellen auf und ist anfangs leer. Auf jeder Zelle kann maximal eine Kohlpflanze wachsen. Eine Kohlpflanze soll eine abstrakte Maximalgröße von 15 haben und einen Reifegrad von 0 bis 5.
Unsere Kohlsorte wächst zunächst bis zu seiner Maximalgröße. Erst jetzt beginnt der Reifeprozess und der Kohl reift bis zum maximalen Reifegrad (ja, diese Kohlsorte muss tatsächlich reifen :-). Überschreitet eine Kohlpflanze den maximalen Reifegrad von 5 verrottet (verschwindet) diese, kann also nicht mehr geerntet werden. Durch das Ernten von reifem Kohl können nun Punkte gesammelt werden. Ein hoher Reifegrad bedeutet einen hohen Verkaufserlös für den geernteten Kohl, der sich in einem Punkt pro Reifegrad des geernteten Kohls widerspiegelt. Nicht ausgewachsener oder unreifer Kohl kann zwar geerntet werden, ist aber am Markt unverkäuflich und gibt daher keine Punkte.
Die Ernte der Pflanzen geschieht mit einem Traktor der immer genau eine Zelle belegt. Der Traktor ist zu jedem Zeitpunkt auf einer Rasterzelle platziert und kann mit den Pfeiltasten bewegt werden. Die Ernte-Taste "H" soll den Traktor in den Erntemodus versetzen, so dass dieser den Kohl auf der aktuellen Zelle erntet. Der Traktor soll beim Ernten weiterfahren können und dabei die überfahrenen Zellen abernten. Die Taste "P" soll den Traktor in den Pflanzmodus versetzen. Im Pflanzmodus setzt der Traktor auf allen überfahrenen Feldern, die zuvor leer waren, Setzlinge der Größe 1.
Zeigen Sie die bisher gesammelten Punkte auf dem Bildschirm an. Sorgen Sie dafür, dass nicht alle Pflanzen gleichmäßig schnell wachsen und dass die Zeiteinstellungen und die grafische Darstellung der Objekte so sind, dass das Spiel bedienbar ist.
Tipp: Mit der Klasse `java.util.Random` können Sie das Wachstum der Pflanzen mit einer Zufallskomponente versehen.

Aufgabe 3 (Command-Pattern und Undo, 4 Punkte)
Projekt: `ex04_3`. Package: `undo`.
Objektorientiertes Design besteht nicht immer nur daraus, greifbare Dinge wie Lastwagen oder Schwerter mit Objekten nachzubauen. Manchmal ist es auch sinnvoll, Verben zu Substantiven zu machen und Begriffe wie "das Eintragen des Wertes 5 in die Zelle A1 der Tabelle" mit einem Objekt zu repräsentieren. Wir werden das nun mit einer ganz kleinen Tabellenkalkulation mit Undo-Funktion durchspielen.

(a) Bauen Sie zunächst eine Klasse `Sheet`, die eine (eindimensionale) Tabelle von `int`-Werten repräsentiert. Man soll `Sheet` mit der Anzahl Zellen instanziieren können. `Sheet` soll Methoden haben, um in eine Zelle einen Wert hineinzuschreiben (`void put(int index, int value)`), und aus einer Zelle den Wert auszulesen (`int get(int index)`). Bei ungültigen Indizes darf die Klasse sich beliebig verhalten.
Außerdem soll `Sheet` die Inhalte aller Zellen, mit einem Trennzeichen getrennt, in einen String umwandeln können (`String toString()`). Das Trennzeichen soll im `Sheet` als Feld gespeichert werden, mit einer `getSeparator`- und einer `setSeparator`-Methode. Tipp: `StringBuilder` kann hilfreich sein..
Testen Sie `Sheet`.

(b) Das Schreiben eines bestimmten Wertes in eine bestimmte Zelle soll durch die Klasse `PutCommand` modelliert werden. Sie soll das Interface `ICommand` implementieren:

```
1  public interface ICommand {
2    /** Perform the modification represented by this object on a sheet.
3     * @param sheet the sheet to modify.
4     */
5    public void executeOn(Sheet sheet);
6  }
```

Wenn man nun irgendwo

```
1  ICommand cmd = new PutCommand(2, 23);
2  cmd.executeOn(sheet);
```

ausführt, soll in `sheet` in Zelle 2 der Wert 23 eingetragen worden sein.
Testen Sie `PutCommand`.

(c) Analog zu `PutCommand` soll es eine Klasse `SetSeparatorCommand` geben. Sie modelliert "das Setzen des Trennzeichens auf einen bestimmten Wert".
Testen Sie auch `SetSeparatorCommand`.

(d) Das Tabellenkalkulationsprogramm ist eine Klasse `SpreadSheetApp`.

```
1  public class SpreadSheetApp {
2    public SpreadSheetApp(Sheet s) {...}
3    public void put(int cell, int value) {...}
4    public void setSeparator(char sep) {...}
5    public void undo() {...}
6    /** @return the sheet's string representation */
7    public String display() {...}
8  }
```

Sie lässt den Zugriff auf ihr inneres `Sheet` nur über Methoden `put` und `setSeparator` zu, die vorher auf geeignete Weise Rückgängigmachungs-Information aufbewahren. Implementieren Sie die Klasse. Finden Sie einen Weg, die Undo-Funktionalität anzubieten, und nutzen Sie dabei die Command-Klassen (das wäre doch sonst Verschwendung).
Der Einfachheit halber braucht Ihre Klasse nur einstufiges Undo zu unterstützen; was passiert, wenn mehrmals direkt hintereinander `undo()` aufgerufen wird, ist Ihnen überlassen.
Testen Sie die Undo-Funktionalität.

Aufgabe 4 (Verständlichkeit und Schwierigkeitsgrad, 1 Punkt)
Projekt: `ex04_4`. Package: `difficulty`.
In dieser Aufgabe sollen Sie ein Programm schreiben, was in einigen Sätzen auf dem Bildschirm ausgibt, wie Sie bisher mit dem Stoff aus der Vorlesung und dem Schwierigkeitsgrad der Übungsaufgaben zurechtgekommen sind.

Nachholaufgaben für Thema 2: Einfache Klassen

Mit dieser Aufgabe können Sie den Stoff von Thema 2 (dem Thema von Blatt 2) wiederholen und nochmal Punkte für Thema 2 erwerben.

Aufgabe 5 (Putzkolonnen, 4 Punkte)
Projekt: `ex02_4` Package: `verwalt`
Aus der Anforderungsspezifikation einer Verwaltung für Reinigungstrupps:

> Mitarbeiter haben einen Namen und einen Wohnort (gegeben als Adresse). Sie arbeiten an einem Einsatzort (gegeben als Adresse) und holen ihr Werkzeug täglich in einem der Regionallager (gegeben als Adresse). Mitarbeiter mit mehr als 100km Luftlinie-Entfernung zwischen Wohnort und Einsatzort, Einsatzort und Materiallager oder Materiallager und Wohnort haben Anspruch auf Entfernungszulage.

(a) Extrahieren Sie sinnvolle Klassen und implementieren Sie sie. Es muss möglich sein, ein Mitarbeiter-Objekt zu fragen, wie der Mitarbeiter heißt, was seine Wohn-/Einsatz-/Lager-Orte sind und ob der Mitarbeiter Entfernungszulage beanspruchen kann. Zunächst können Sie annehmen, dass die Luftlinie-Entfernungen extern von Hand berechnet und bei der Erzeugung der Objekte eingegeben werden.
Testen Sie nichttriviale Methoden.

(b) Schauen Sie (z.B. in der Wikipedia) nach, wie man die Entfernung zwischen zwei Orten berechnet (Stichwort: Großkreis), die nur per Längen- und Breitengrad gegeben sind. Bauen Sie Ihre Klassen so um, dass Entfernungen aus den Koordinaten der Adressen berechnet werden. Welche Klasse sollte für das Berechnen von Entfernungen verantwortlich sein? Begründen Sie kurz im Klassenkommentar.
Testen Sie die neue Entfernungsberechnung.