P. Thiemann und K. Anton Sommersemester 2010

Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2010/

Betreutes Java-Programmieren 5

2010-05-31

Anfang Loggen Sie sich in der normalen Linux-Umgebung ein.

Starten und Speichern Starten Sie Eclipse, indem Sie in einem Terminalfenster `eclipse` eingeben. Wenn Eclipse Sie fragt, wo der Workspace hin soll, antworten Sie bitte `/home/`*`ihrloginname`*`/workspace`, was normalerweise auch schon voreingestellt sein sollte.

Drumherum Unter der URL `http://nonopapa:8080/teaching/java/2010/` steht Ihnen das Vorlesungsmaterial zur Verfügung. Die Sun-Java-Doku gibt es unter `http://nonopapa:8080/javadoc/`

Bei Fragen zur Rechnerbenutzung, zu den Aufgaben oder zu diesen Hinweisen wenden Sie sich bitte sofort an einen der Tutoren.

Ende Erzeugen Sie ein ZIP- oder tgz-Archiv Ihres Eclipse-Projects: Rechtsklick aufs Project, "Export...", im Ast "General" die Zeile "Archive File" auswählen, prüfen, dass links oben genau Ihr Project angekreuzt ist, im Feld "To archive file" einen Dateinamen eingeben, der Ihren Namen enthaelt: zB `/home/mustermm/Desktop/abgabe3-mustermm.zip` für Max Mustermann, "Finish" klicken. Dabei sollte ein ZIP-Archiv enstehen, das Sie am Ende dem Tutor mailen. Nur ein Attachment pro Mail!

Dieses Blatt geben Sie bitte den Tutoren zurück.

Eclipse-Project. Heute brauchen Sie das Eclipse-Projekt nicht selbst anzulegen – Sie starten mit einem halbfertigen Projekt. Laden Sie zunächst `http://nonopapa:8080/teaching/java/2010/supp/skel5-life.zip` herunter (skel wie "Skelett"). Dann Kontextmenü im Package-Explorer aufmachen, "Import...", "General" aufklappen, "Existing projects into workspace" auswählen, "Next" klicken, oben "Select Archive File" auswählen, "Browse" drücken, das runtergeladene `skel5-life.zip` suchen, dann "Finish" klicken. Sie sollten jetzt ein Project "abgabe5" haben.

Life. Ein hübscher zellulärer Automat ist Life[1] . Es gibt ein rechteckiges Gitter von Zellen. Zellen sind tot oder lebendig. Wenn von den acht Nachbarn einer toten Zelle genau drei lebendig sind, entsteht dort eine lebende. Eine lebendige Zelle überlebt nur, wenn von ihren acht Nachbarn zwei oder drei lebendig sind (sonst vereinsamt oder erstickt sie). Damit jede Zelle acht Nachbarn hat, ist das Gitter zyklisch: der linke Nachbar der linkesten Spalte ist die rechteste Spalte etc. Koordinaten fangen bei 0 an. Die Regeln werden schrittweise auf alle Zellen gleichzeitig angewendet: ob in der $n+1$. Generation an einer Stelle eine lebendige Zelle liegt, kommt auf den Zustand der Nachbarzellen in der n. Generation an.

```
.....     ..O..       ..O..     .....
.OOO. --> ..O..       .OOO. --> .O.O.
.....     ..O..       ..O..     .....
```

Linkes Beispiel: die äußeren Zellen sterben an Vereinsamung, die mittlere überlebt mit 2 Nachbarn, und über und unter der mittleren Zelle entstehen neue. Rechtes Beispiel: die drei in der Mitte ersticken (je vier Nachbarn, dank des zyklischen Gitters), die zwei rechts und links überleben dank dreier Nachbarn. **Package:** `life`.

Aufgabe 1 (Model, 4+3+6+2 Punkte)
Es ist meist sinnvoll, das Modell getrennt von der Benutzeroberfläche zu entwickeln. So kann man die Berechnungen in Ruhe testen, ohne sich um Farbe und Fenstergröße zu kümmern. Das Model-Interface `IGridModel` steht schon bereit. Wir werden intern das Gitter durch ein zweidimensionales Array von `int` repräsentieren, das 0 für "tot" und 1 für "lebendig" enthält[2].

[1] na-naa-na-na-naa.

[2] Oder man nimmt `boolean`, aber dann ist die 1c mühsamer.

(a) Schreiben Sie eine Klasse `LifeGrid`, die `IGridModel` implementiert. Der Konstruktor soll mit `new LifeGrid(numRows, numColumns)` aufgerufen werden können. Schreiben Sie zunächst Testfälle für `getRows`, `getColumns` und das Paar `setCell-getCell` und implementieren Sie diese Methoden dann fertig[3]. Die Tests sollen entdecken, ob Sie irgendwo Zeilen und Spalten verwechselt haben.

(b) Statten Sie `LifeGrid` mit einer Methode `String toString()` aus, die eine Stringdarstellung des Gitters wie in den Beispielen oben erzeugt. Tote Zellen sollen durch Punkte, lebendige durch den Großbuchstaben O dargestellt werden. Jede Zeile soll durch ein Newline abgeschlossen werden, das Sie mit `'\n'` erzeugen können. Zwei Tests.

(c) Erwecken wir die Zellen zum Leben! Schreiben Sie zunächst zwei Testfälle, in denen Sie einige Zellen plazieren, `computeNextGeneration()` aufrufen und das Ergebnis von `toString()` mit einem Erwartungswert vergleichen. Implementieren Sie dann `computeNextGeneration()`. Sie werden wohl intern zwei Arrays brauchen (für alte und neue Generation). Für Randfälle lohnt sich ein Blick in `WraparoundDemo`. Vielleicht ist eine (testbare) Hilfsmethode `countNeighbours(int row, int col)` sinnvoll?

(d) Leere Gitter sind langweilig. Implementieren Sie eine Methode `randomize()`, die jede Zelle zufällig auf tot oder lebendig setzt. Die Klasse `java.util.Random` kann da helfen, insbesondere die Methode `nextInt(int)`. Kein Test.

Aufgabe 2 (View, 2+3 Punkte)
Zur Visualisierung finden Sie im Skelett schon eine Klasse `AbstractLifeApp` und eine Klasse `AbstractLifeView`. Dort sind die nervigen Details der Swing-Programmierung schon weitgehend abgehakt, nur das Zeichnen der Zellen fehlt.

(a) Erzeugen Sie eine Subklasse `LifeView` von `AbstractLifeView` und eine Subklasse `LifeApp` von `AbstractLifeApp`. Die Methode zum Zeichnen der Zellen können Sie zunächst leer lassen. Die Klasse `GuiRunner` hat eine `main`-Methode – bringen Sie damit ein graues Fenster auf den Schirm.

(b) Implementieren Sie die `paintCells`-Methode, so dass die Zellen angezeigt werden. `Graphics` hat dazu Methoden mit Namen wie `setColor` und `fillRect`.

Praktische Eclipse-Tastenkombinationen

- Ctrl+Space: den Identifier unterm Cursor vervollständigen
- Ctrl+1: Quick Fix (leichtere Reparaturen automatisch vornehmen, z.B. alle fehlenden Methoden des Interfaces implementieren)
- F3: gehe zur Definition der Klasse/Methode/... unterm Cursor
- Ctrl+Shift+T: gehe zu Klasse, gegeben die Grossbuchstaben des Klassennamens
- Ctrl+Shift+G: liste alle Verwendungen der Klasse/Methode/... unterm Cursor auf
- Alt+Shift+J: kommentiere das markierte Sprachelement
- Alt+Shift+S: diverse Source-Transformationen, z.B. "Generate Constructor using Fields"
- Alt+Shift+R: Rename
- Und einfach mal das Source- und das Refactor-Menü durchstöbern, man lernt eigentlich immer was dazu.

[3] Unfertige Methoden kann man mit `throw new UnsupportedOperationException();` auffüllen, damit sie kompilieren.