

Dozenten

Prof. Dr. Thomas Vetter
Dr. Marcel Lüthi
Departement
Mathematik und Informatik
Spiegelgasse 1
CH – 4051 Basel

Assistent

Dr. Marcel Lüthi
Departement
Mathematik und Informatik
Spiegelgasse 1
CH – 4051 Basel

Tutoren / Tutorinnen

Claudia Grundke
Viktor Gsteiger
Simon Dold
Timo Steinebrunner
Alexander Rovner
Nikodem Kernbach
Lukas Stöckli

Erweiterte Grundlagen der Programmierung (45398-01)**Blatt 10****[10 Punkte]**

Vorbesprechung 25. November - 29. November

Abgabe 06. Dezember

Allgemeine Hinweise

- Wir empfehlen Ihnen, dass Sie im Buch “Sprechen Sie Java” bis und mit Kapitel 19 lesen.
- Schauen Sie sich auch die Vorlesungsslides zum Thema AWT an.

Voraussetzung

- Es gelten dieselben Voraussetzungen wie für Übungsblatt 1. Wenn Sie sich betreffend der Umgebung oder der automatisierten Tests noch unsicher sind, lesen Sie bitte nochmals sorgfältig das Infoblatt oder fragen Sie die Tutoren.
- Die Zip-Datei, die auch dieses Übungsblatt enthält, muss entpackt werden. Es enthält die gesamte Übungsumgebung inklusive der automatisierten Tests. Schreiben Sie Ihre Lösungen in die dafür vorgesehenen Dateien, wie in der jeweiligen Übungsaufgabe angegeben.

Empfohlenes Vorgehen

- Wechseln Sie in den Ordner `src/main/java`. Dort finden Sie die Dateien, in welche Sie Ihren Java Code schreiben.
- Schreiben Sie Ihr Programm, kompilieren Sie dieses mit dem Java Compiler `javac` und führen Sie es mit `java` aus, wie es in der Vorlesung gezeigt wurde.
- Wenn Sie denken, dass alles in Ordnung ist, wechseln Sie zurück ins Übungsverzeichnis `uebung10` und führen da `gradlew test` aus, um zu überprüfen ob Ihre Lösung die automatisierten Tests besteht. Überprüfen Sie auch Ihren Codestil mit `gradlew checkstyleMain`. Falls Ihr Code den Vorgaben entspricht, erhalten Sie einen Bonuspunkt.

Abgabe

Ergänzen Sie die Datei `email.txt` mit Ihrer Unibas E-Mail Adresse. Erstellen Sie eine Zip-Datei der gesamten Übungsumgebung (also des Verzeichnisses `uebung10`) und laden Sie dieses auf Courses (<https://courses.cs.unibas.ch/>) hoch.

Aufgabe 1 - Exceptions

[2 Punkte]

Sie finden im Verzeichnis `src/main/java/list` die Klasse `LinkedList`. Implementieren Sie die Methode `removeFirst`, welche das letzte Element der Liste entfernt und zurückgibt. Schreiben Sie eine Klasse `ListException` (Achtung, Exception Klassen müssen `public` sein). Wenn die Methode `removeFirst` auf einer leeren Liste aufgerufen wird, soll diese eine `ListException` mit der Meldung "remove on empty list" ausgegeben.

Implementieren Sie nun die Methode `removeAll`, welche alle Elemente der Liste entfernt, indem mehrmals die Methode `removeFirst` aufgerufen wird. Die Methode `removeAll` darf keine Exception werfen.

Aufgabe 2 - Game of Life

[4 Punkte]

Das Spiel des Lebens (Game of Life) geht auf den Mathematiker John Conway zurück und funktioniert nach folgenden Prinzipien:

(<http://www.math.com/students/wonders/life/life.html>)

Die Grundeinheit sind Zellen, die in einer Matrix angeordnet sind. Jede Zelle kann lebendig oder tot sein. Jede Zelle hat acht Nachbarn, wobei Randzellen die Zellen des gegenüberliegenden Randes als Nachbarn haben. Der Zustand der Zellen (lebendig oder tot) ändert sich von Generation zu Generation. Die aktuelle Zellpopulation beeinflusst die darauf folgende Generation nach folgenden Regeln:

- (a) Eine tote Zelle mit genau drei lebenden Nachbarn erwacht zum Leben (birth).
- (b) Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt am Leben (survival).
- (c) Alle anderen lebenden Zellen sterben (overcrowding or loneliness).

Sie finden im Verzeichnis `src/main/java/gol` die Klasse `GameOfLife` welche als Feld eine Zellenpopulation als eine $size \times size$ Matrix vom Typ `boolean` speichert. Jedes Element hat entweder den Wert `false` (tote Zelle, '.') oder `true` (lebende Zelle, '@'). Speichern Sie auch die Größe `size` als Feld der Klasse.

Konstrukturen

[2 Punkte]

Implementieren Sie den Konstruktor, welcher ein `boolean` Array mit einem gegebenen Muster (repräsentiert als zweidimensionales `boolean` Array) entgegennimmt und die entsprechenden Felder setzt.

Danach implementieren Sie die statischen Methoden `createBlock`, `createBlinker` und `createGlider`, welche ein neues Game of life mit folgenden Mustern initialisiert:

```
....  
.@@.  
.@@.  
....  
(Block)
```

```
.....
..@..
..@..
..@..
.....
(Blinker)
```

```
.....
.@.@..
..@@..
..@...
.....
.....
(Glider)
```

Implementieren Sie ausserdem eine statische Methode `createRandom` welches ein Objekt der Klasse `GameOfLife` für eine angegebene Feldgrösse erstellt. Der Wert jeder Zelle soll dabei zufällig gesetzt werden. Dabei wird jede Zelle mit der angegebenen Wahrscheinlichkeit als "lebend" initialisiert. Verwenden Sie dazu die Klasse `java.util.Random`.

Evolution

[1 Punkt]

Implementieren Sie die Methoden `isActive`, die den Wert aus dem Feld abzufragen, eine Methode `getNumberOfActiveNeighbors` welche die lebenden Nachbarn zählt, und eine Methode `update` welche das Update ausführt. Dabei soll der Zustand aller Zellen zur "gleichen" Zeit ausgeführt werden. Verwenden Sie dazu eine Kopie der Zellpopulation. Beachten Sie ferner dass der Zugriff auf eine Zelle jeweils auf der gegenüberliegenden Seite geschieht wenn auf die Nachbarn von Zellen am Rand zugegriffen wird.

Ausgabe

[½ Punkt]

Schreiben Sie nun noch die Methode `public String toString()` welche die Population in einem String darstellt. Um Zeilenumbrüche in einem String darzustellen können Sie die Zeichenfolge `"\n"` verwenden. Jeder Zelle soll dabei durch ein `.` oder ein `@` dargestellt werden.

Testprogramm

[½ Punkt]

Implementieren Sie die Methoden im Testprogramm `GameOfLifeCommandLine`, welches die Evolutionsschritte vom `GameOfLife` berechnet und das Ergebnis jeweils auf der Konsole ausgibt.

Aufgabe 3 - Game of Life - AWT

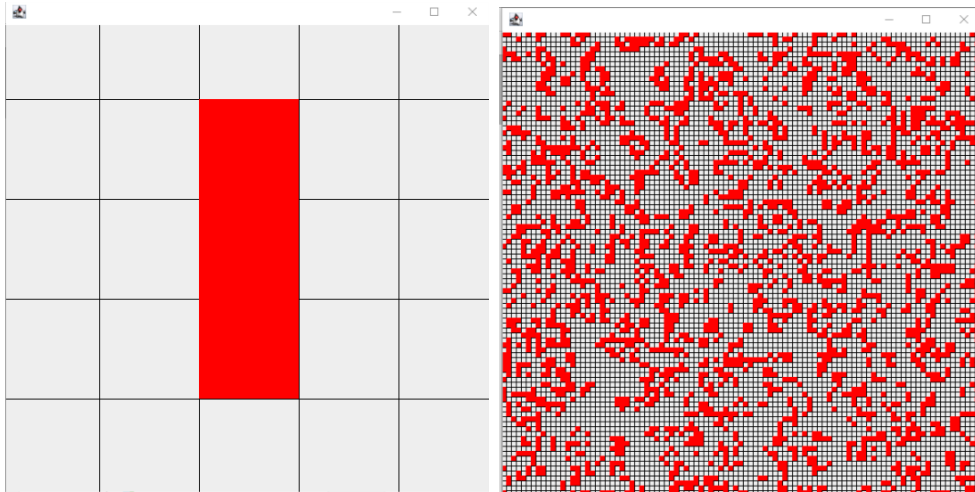
[2 Punkte]

Hinweis: Falls Sie die vorige Aufgabe nicht lösen konnten, können Sie Klasse `MockGameOfLife` nutzen, die Sie im Verzeichnis `src/main/java/gol/` finden.

Sie finden im Verzeichnis `src/main/java/gol` die Java-Klasse `GOLWindow` zur Visualisierung des Game Of Life. Implementieren Sie die fehlenden Methoden in dieser Klasse, gemäss den angegebenen Spezifikationen in den Kommentaren.

Experimentieren Sie mit verschiedenen Konfigurationen. Für die Konfiguration `Blinker` (und das `MockGameOfLife`), sollten ihre Ausgabe im ersten Schritt etwas so wie das Bild

links aussehen. Wenn Sie grosse, zufällige Welten simulieren, sieht die Ausgabe etwa wie im Bild rechts aus.



Aufgabe 4 - Game of Life - AWT mit Double Buffering

[2 Punkte]

Wenn Sie die vorige Übung gelöst haben ist ihnen vielleicht aufgefallen, dass bei grossen Welten die Darstellung ins Stocken gerät. Abhilfe schafft hier das *Double buffering*. Double buffering ist eine Strategie, bei welcher nicht direkt auf den Screen gezeichnet wird, sondern erst wird ein Bild gezeichnet, welches dann in der Paint Methode jeweils direkt dargestellt wird. Die Klasse `JFrame` (respektive deren Superklasse `Component`) stellt dafür bereits die Methode `createImage` zur Verfügung, welches ein Offscreen Image erstellt, in welches gezeichnet werden kann.

Schreiben Sie ihr Programm nun so um, dass es double buffering benutzt. Dazu kopieren Sie ihre Lösungen zur vorigen Aufgabe erst in die Klasse `GOLWindowDoubleBuffering` und passen diese dann entsprechend an. Erstellen Sie sich ein Feld `offscreenImage` sowie ein Feld `offscreenGraphics`. Das Feld `offscreenGraphics` ist vom Datentyp `java.awt.Graphics`. Beim ersten Aufruf der Paint-Methode soll nun mittels `createImage` das `offscreenImage` gesetzt werden und mit dem Befehl `offscreenImage.getGraphics()` das Feld `offscreenGraphics`. Wenn diese Felder gesetzt sind, zeichnen Sie nun in dieses Bild und nutzen dann die Methode `drawImage` um das offscreen gerenderte Image darzustellen (Achtung: stellen Sie sicher, dass Sie das richtige Graphics-Objekt nutzen.)

Hinweis: Sie dürfen bei dieser Teilaufgabe auch im Internet recherchieren um Codebeispiele zu finden.