Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Jens-Peter Redlich
M. Kabierski, A. Heuts, Dr. W. Müller, M. Sänger

Berlin, den 07.12.2020

Übungen zur Vorlesung Grundlagen der Programmierung

Übungsblatt 3

Abgabe: Die Abgabe Ihrer Lösungen erfolgt über den Moodle-Kurs der Vorlesung (https://moodle.hu-berlin.de/course/view.php?id=98193). Nutzen Sie die dort im Abschnitt Übung angegebene **Aufgabe 3.** Die Abgabe ist bis zum **04.01.2021** um **09:15 Uhr** möglich. **Die Abgabe erfolgt in Gruppen.** Bitte bilden Sie im Moodle Gruppen! Bitte verwenden Sie **keine** (noch nicht in der Vorlesung eingeführten) **Java-Bibliotheken**.

Hinweise:

- Achten Sie darauf, dass Ihre Java-Programmdateien mittels des UTF-8-Formats (ohne byte order marker (BOM)) kodiert sind und keinerlei Formatierungen enthalten.
- Verzichten Sie auf eigene Packages bei der Erstellung Ihrer Lösungen, d.h. **kein** `package`-Statement am Beginn Ihrer Java-Dateien.
- Quelltextkommentare können die Korrektoren beim Verständnis Ihrer Lösung (insbesondere bei inkorrekten Abgaben) unterstützen; sind aber nicht verpflichtend.
- Testen Sie Ihre Lösung bitte auf einem Rechner aus dem Informatik Computer-Pool z.B. gruenau6.informatik.hu-berlin.de (siehe auch Praktikumsordnung: Referenzplattform)

Aufgabe 1 (Sieb des Eratosthenes) 10 Punkte

Mit dem aus der Vorlesung bekannten Algorithmus *Sieb des Eratosthenes* kann für jede ganze Zahl zwischen 2 und einer Grenze M bestimmt werden, ob es sich um eine Primzahl handelt. Ihre Aufgabe ist es, die aus der VL gegebene Implementierung zu erweitern, und ein Java-Programm `Sieb.java` zu schreiben, welches die ersten N Primzahlen ausgibt. Der Parameter **N** soll Ihrem Programm als Kommandozeilenparameter übergeben werden:

```
$ java Sieb <N>
```

Weiterhin soll jede Primzahl in einer eigenen Zeile ausgegeben werden. Ein Aufruf des Programms, der die ersten 5 Primzahlen ausgibt sieht somit folgendermaßen aus:

```
$ java Sieb 5
2
3
5
7
11
```

Beachten Sie, dass die aus der VL gegebene Implementierung „lediglich“ alle Primzahlen im Bereich von 2 bis M berechnet. Bei der Wahl einer bestimmten Grenze M kann es also vorkommen, dass der Algorithmus weniger als N Primzahlen findet. In diesem Fall soll die Grenze erhöht und die gesamte Berechnung erneut durchgeführt werden. Starten Sie z.B. mit einer Grenze $\mathrm{M} = 9$ für das Sieb und erhöhen Sie die Grenze nach jedem Durchlauf um Faktor 10.

Aufgabe 2 (Game of Life) 10 Punkte

Conways *Game of Life* ist ein einfaches Spiel auf einem zweidimensionalen Feld, das mit Hilfe von vier Regeln das Leben, Sterben und Entstehen von Zellen simuliert. Das komplette Feld besteht ausschließlich aus rechteckigen Zellen, die zu einem festen Zeitpunkt *t* entweder leben oder tot sind. Ob eine Zelle zu einem Zeitpunkt t+1 tot oder lebendig ist, hängt von der direkten Nachbarschaft der Zelle im vorherigen Zeitpunkt t ab. Jede Zelle hat genau acht direkte Nachbarn:

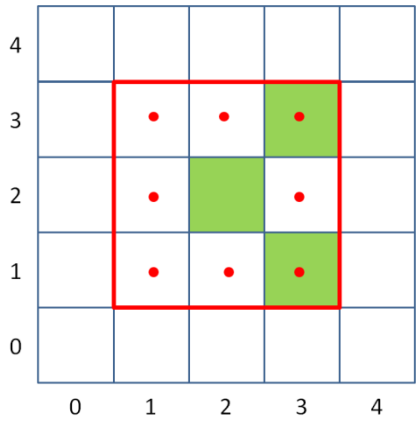

Abbildung 1: Nachbarn von [2][2] auf Beispielfeld 5x5. Grüne Felder stellen aktuell lebende Zellen dar.

In Abbildung 1 hat die aktuell lebende Zelle mit der Koordinate [2][2] alle rot markierten Zellen als direkte Nachbarn. Davon sind zudem zwei ([3][1]

und [3][3]) ebenfalls am Leben. Um zu ermitteln ob eine Zelle im Zeitpunkt *t+1* lebt oder nicht, gilt es folgende Regeln im Moment *t* auszuwerten:

1. Ist die Zelle tot und hat exakt drei lebende Nachbarn, wird sie lebendig.
2. Lebt die Zelle bereits und hat 2 oder 3 lebende Nachbarn, bleibt sie am Leben.
3. In allen anderen Fällen stirbt sie.

Für das Spielfeld in Abbildung 1 ergibt sich also im Folgezustand, dass lediglich die Zellen mit Koordinaten [2][2] und [3][2] lebendig sind.

Felder am Rand des Spielfelds haben weniger als 8 Nachbarn. Nennen Sie in diesen Fällen an, dass der obere Rand mit dem unteren und der linke Rand mit dem rechten verbunden ist. Dies bedeutet, dass in dem gegebenen Beispiel jede Zelle mit einer x-Koordinate von 4 drei rechte Nachbarn mit den x-Koordinaten 0 hat. Entsprechend hat jede Zelle am oberen Rand drei obere Nachbarn mit den y-Koordinaten 0. In den vier Eckpunkten gelten beide Überläufe. Die Abbildungen 1 und 2 illustrieren das Vorgehen grafisch.

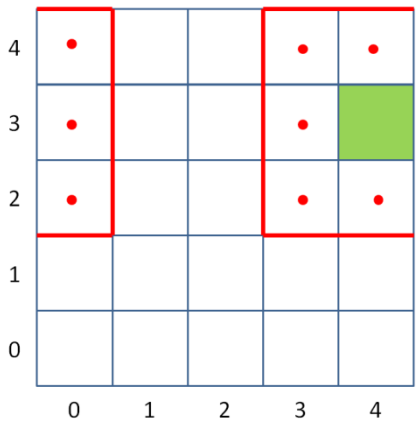

Abbildung 2: Nachbarn von [4][3]

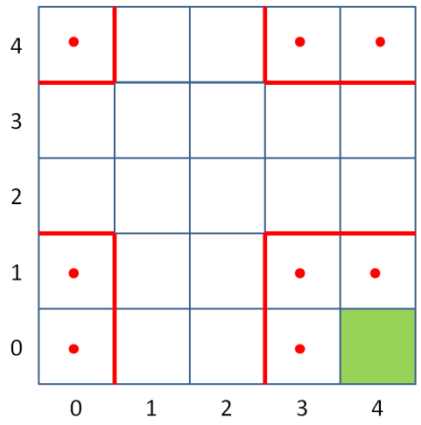

Abbildung 3: Nachbarn von [4][0]

Schreiben Sie ein Java-Programm `GameOfLife.java`, das eine Startkonfiguration für ein Spielfeld einliest und anschließend eine Simulation startet, die in einer Endlosschleife den aktuellen Zustand des Spielfeldes graphisch ausgibt, 200ms wartet und anschließend den Folgezustand berechnet.

Die Daten für die Startkonfiguration werden in dieser Aufgabe nicht als Argumente beim Programmaufruf übergeben, sondern sollen von der Standardeingabe gelesen werden sobald das Programm gestartet wurde. Verwenden Sie hierfür die in der Vorlesung vorgestellte Klasse `StdIn` aus der Bibliothek `gdp_stdlib.jar`, die Ihnen als Ressource zur Verfügung

gestellt wird.

Jede Konfiguration besteht aus mindestens drei Zahlen am Anfang, die Sie mit `StdIn.readInt()` einlesen sollen. Die ersten beiden Zahlen beschreiben die Größe der x-Achse und der y-Achse des Spielfeldes, das verwendet werden soll. Die dritte Zahl definiert die Anzahl der lebenden Zellen. Danach folgen die Koordinaten der lebenden Zellen als Paare von Zahlen.

Im beigefügten Ordner GoLRessources finden Sie vier einfache Startkonfigurationen zum Testen Ihres Programms.
Beispielsweise sehen die ersten 5 Zeilen der Datei GOLSpaceships.conf folgendermaßen aus:

```
20 20 14

1 1
1 3
2 4
… …
```

Die erste Zahl (`20`) gibt die Größe der x-Achse des Spielfelds an, während die zweite Zahl (`20`) die Größe der y-Achse des Spielfelds angibt. Es folgt die Anzahl an anfangs lebendigen Zellen, hier 14. In den darauffolgenden Zeilen, folgen pro Zeile Paare von x-und-y-Koordinaten für die 14 lebenden Zellen. Die ersten 3 angegebenen Zellen haben hier also die Koordinaten `[1][1]`, `[1][3]` und `[2][4]`.

Um den Inhalt einer Datei in den Eingabestrom umzuleiten, können Sie beispielsweise folgenden Befehl auf der Kommandozeile ausführen:

```
java -cp .:gdp_stdlib.jar GameOfLife < GOLSpaceships.conf
```

In Eclipse können Sie unter *Run Configurations* > *Commons* eine Datei als Eingabe für Ihr Programm definieren.

Hinweise zum Zeichnen des Spielfelds
Verwenden Sie zum Zeichnen des aktuellen Spielfeldes die Klasse `StdDraw`. Zeichnen Sie die Linien des Feldes mit der Farbe `StdDraw.BLUE` ein. Lebende Zellen sollen mit `StdDraw.GREEN` und tote Zellen mit `StdDraw.WHITE` gezeichnet werden. Für das Zeichnen des Spielfelds bzw. der Zellen können Sie folgende Funktionen nutzen:

```
StdDraw.filledSquare(double x, double y, double radius)

StdDraw.line(double x0, double y0, double x1, double y1)
```

Mit `StdDraw.show(int t)` zeichnen Sie das Fenster und warten `t Millisekunden`.

Hinweise zum Einbinden der gdp_stdlib.jar
Wenn Sie Ihr Programm per Kommandozeile kompilieren und ausführen, nutzen Sie folgende Befehle:

```
javac -cp .:gdp_stdlib.jar GameOfLife.java

java -cp .:gdp_stdlib.jar GameOfLife
```

Sollten Sie Windows benutzen, so ersetzen Sie die Doppelpunkte durch ein Semikolon.
In Eclipse können Sie unter *Project > Properties > Java Build Path > Libraries > Classpath > Add External JARs* die `gdp_stdlib.jar` zu Ihrem Projekt hinzufügen.

Tipps zur Bearbeitung
Die Entwicklung des Programms erfordert die Lösung verschiedener Teilaufgaben (u.a. das Einlesen der Startkonfiguration, die Berechnung des nächsten Zustands sowie die Anzeige des Spielfelds). Bei der Erarbeitung Ihrer Lösung kann es ggf. hilfreich sein sich Schritt-für-Schritt diesen einzelnen Teilaufgaben zu widmen, ohne sofort die endgültige Gesamtlösung zu erdenken.

Viel Spaß und Erfolg bei der Lösung der Aufgaben!