

Aufgabenblatt 02

Ziel dieses Aufgabenblatts ist es, Sie weiter mit der Anwendung von Interfaces vertraut zu machen, um eine lose Kopplung zwischen Klassen unterschiedlicher Packages sicherzustellen und Abhängigkeiten umzukehren.

Abgabe: Gr. 1: 04.11.2022, Gr. 2: 16.11.2022, Gr. 3: 14.11.2022; Max. Punktzahl: 20; Min. Punktzahl: 14 Punkte

Aufgabe 2: „Spiel-des-Lebens“ von Conway (20 Punkte)

In dieser Aufgabe entwickeln Sie mit der Programmiersprache Java eine Simulation von Conways „Spiel des Lebens“. Das Spielfeld ist in Zeilen und Spalten unterteilt. Jedes Quadrat ist ein zellulärer Automat, der einen von zwei Zuständen, konkret „bewohnt“ und „unbewohnt“, annehmen kann. Wir setzen hierfür auf Ebene der Logik ein zweidimensionales `boolean`-Array ein (`boolean[][]`).

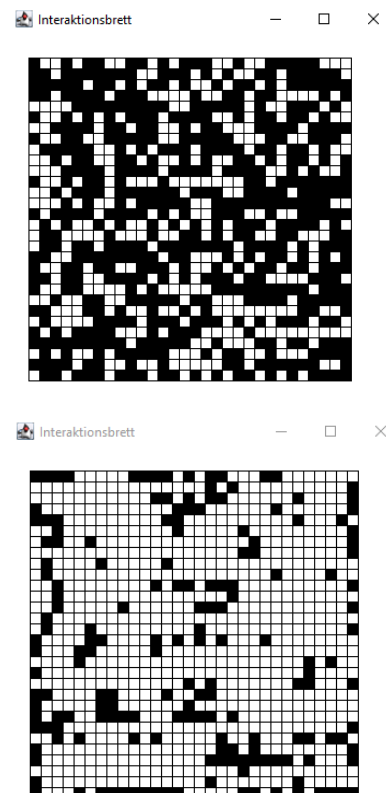
Zu Beginn des Spiels wird eine Anfangsgeneration von lebenden Zellen auf dem Spielfeld platziert. Hierzu kann der Nutzer nach dem Start über die Kommandozeile die Spielfeldgröße, sowie einen Wahrscheinlichkeitswert zwischen 1 und 100 angeben. Sie können zur Mensch-Maschine-Interaktion die beigefügte Java-Klasse `EinUndAusgabe.java` Ihrem Projekt in das Package `de.hsos.prog3.ab02.util` hinzufügen und verwenden.

Der initial durch den Nutzer definierte Wahrscheinlichkeitswert wird verwendet, um zu entscheiden, ob eine Zelle bewohnt ist oder nicht. Hierzu wird für jede Zelle ein Zufallswert über die Klasse `Random` ermittelt. Ist dieser Zufallswert kleiner, als der definierte Wahrscheinlichkeitswert, ist die Zelle bewohnt, ansonsten unbewohnt. Je größer der Wahrscheinlichkeitswert gewählt wird, desto mehr Zellen der Anfangsgeneration werden bewohnt sein.

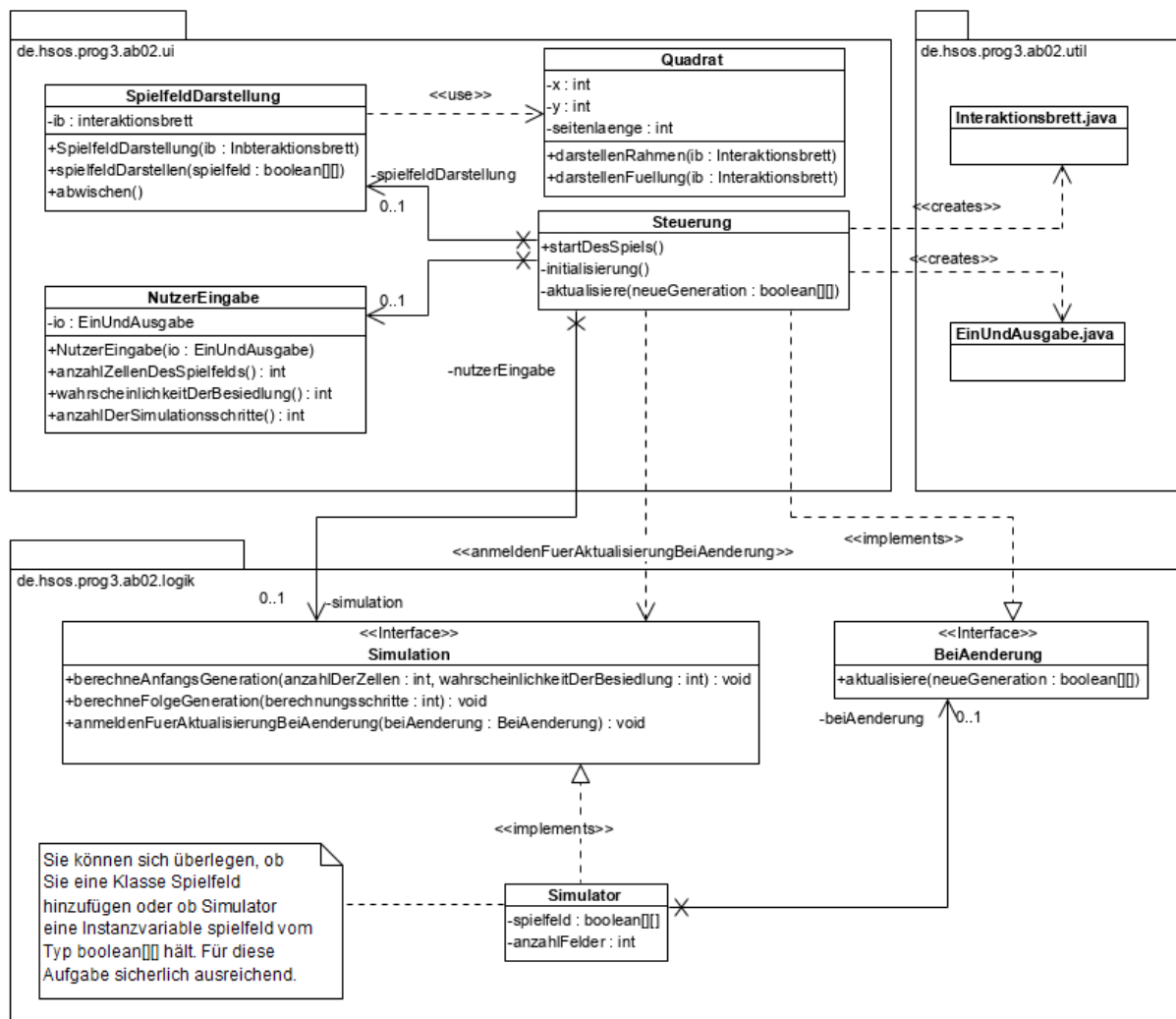
Folgegenerationen lassen sich über die Nachbarschaften einer Zelle anhand folgender konkreter Regeln berechnen: 1. jede bewohnte Zelle mit genau zwei oder genau drei bewohnten Nachbarn bleibt bewohnt, sonst wird sie unbewohnt; 2. jede unbewohnte Zelle mit genau drei bewohnten Nachbarn wird bewohnt, bleibt sonst unbewohnt; 3. jede Zelle hat damit minimal drei (in der Ecke) und maximal acht Nachbarn (in der Mitte).

Zur Darstellung des Spielfeldes mit den aktuellen Generationen soll die beigefügte Klasse `Interaktionsbrett.java` verwendet werden. Fügen Sie diese Klasse Ihrem Projekt wieder zum Package `de.hsos.prog3.ab02.util` hinzu.

Probieren Sie zunächst auf Papier einige Beispiele aus, wie sich die Zellen verändern können. Gibt es z. B. Strukturen, die, solange sich ihr Umfeld nicht ändert, immer bewohnt bleiben?



OO-Design:



Aufgabe 2.1. Implementierung der Simulationslogik (10 Punkte)

Die Spiellogik soll von der Darstellung getrennt werden. Warum? Wahrscheinlich bedarf das User-Interface häufig einer Anpassung, wohingegen die Spiellogik gleichbleiben wird. Mit dieser Trennung wird das Separation-of-Concerns Prinzip umgesetzt.

Um Software-Bausteine voneinander zu trennen, lassen sich in Java u.a. Packages verwenden. Erzeugen Sie das Package `de.hsos.prog3.ab02.logik` und erzeugen dort folgende Interfaces:

BeiAenderung:

```
package de.hsos.prog3.ab02.logik;

public interface BeiAenderung {
    void aktualisiere(boolean[][] neu);
}
```

und Simulation:

```
package de.hsos.prog3.ab02.logik;

public interface Simulation {
    void berechneAnfangsGeneration(int anzahlDerZellen,
        int wahrscheinlichkeitDerBesiedlung);
    void berechneFolgeGeneration(int berechnungsschritte);
    void anmeldenFuerAktualisierungBeiAenderung(BeiAenderung beiAenderung);
}
```

Falls Sie sich fragen, warum das Interface `BeiAenderung` im Package `de.hsos.prog3.ab02.logik` existiert:

- würde dieses Interface nicht eingefügt, müsste ein Objekt der Klasse `Steuerung` ein Objekt der Klasse `Simulation` kennen, um in der `start`-Methode die Anfangsgeneration und während des Programms mehrfach die Berechnung einer neuen Generation anzufordern
- zudem müsste ein Objekt der Klasse `Simulation` das Objekt der Klasse `Steuerung` kennen, um diesem die Ergebnisse der Berechnung einer neuen Generation zu übergeben.
- ⇒ wir hätten zur Compilezeit eine zyklische Abhängigkeit zwischen den Klassen `Steuerung` und `Simulation`, die wir mit Einführung dieses Interfaces auflösen
- ⇒ die Klasse `Steuerung` implementiert die Methode `aktualisiere` des Interfaces, meldet sich zum Programmstart beim `Simulator`-Objekt über die Methode `anmeldenFuerAktualisierungBeiAenderung(this)` an, das `Steuerung`-Objekt weißt das übergebene Objekt der Instanzvariablen `beiAenderung` vom Typ `BeiAenderung` zu und sobald eine neue Generation beim `Steuerung`-Objekt vorliegt, ruft dieses jedesmal die `aktualisiere`-Methode der Instanzvariable `beiAenderung` auf
- ⇒ die Implementierung der `aktualisiere`-Methode der `Steuerung`-Klasse kann dann die neue Generation über das Objekt `SpielfeldDarstellung` visualisieren lassen.

Erzeugen Sie die Klasse `Simulator` als Implementierung des `Simulation`-Interfaces unter Verwendung des Interfaces `BeiAenderung`. Die Implementierungen der Methoden `berechneAnfangsGeneration(...)` und `berechneFolgeGeneration(...)` rufen sobald eine neue Generation vorliegt, die `aktualisiere`-Methode der Instanzvariable `beiAenderung` auf, falls diese nicht `null` ist.

In diesem Beispiel soll auf eine eigene Klasse `Spielfeld` verzichtet werden. Diese Klasse würde ein zweidimensionales `boolean`-Array und die Anzahl der Spielfelder kapseln. Diese Aufgabe übernimmt hier die Klasse `Simulator`. Als OOP-Puristen können Sie die zusätzliche Klasse jedoch gerne einfügen und das Design entsprechend erweitern.

Prüfen Sie die Implementierung eigenständig in einer eigenen Klasse mit einer `main`-Methode.

Aufgabe 2.2. Implementierung der Visualisierung (6 Punkte)

Die Klasse `NutzerEingabe`:

Erstellen Sie im Package `de.hsos.prog3.ab02.ui` die Klasse `NutzerEingabe`. Diese Klasse hat genau eine Instanzvariable `names io` vom Typ `DateiEinUndAusgabe`. Diese Instanzvariable soll niemals `null` sein. Die Klasse bietet folgende Methoden an, die von der Klasse `Steuerung` verwendet werden:

- `int anzahlZellenDesSpielfelds()`: fordert den Nutzer auf, die Anzahl der Zellen des Spielfelds über die Kommandozeile zu definieren und liefert einen `int`-Wert zurück. Dieser darf nicht kleiner als einer von Ihnen definierten Konstanten sein.
- `int wahrscheinlichkeitDerBesiedlung()`: fordert den Nutzer auf, einen Wahrscheinlichkeitswert zwischen 1 und 100 einzugeben, der als `int`-Wert zurückgegeben wird. Dieser darf nicht kleiner 1 und nicht größer 100 sein.
- `int anzahlDerSimulationsschritte()`: fordert den Nutzer auf, einen Wert für die Anzahl der zu berechnenden Simulationsschritte einzugeben. Dieser Wert legt fest, wie viele Generationen berechnet und visualisiert werden, ohne Interaktionsmöglichkeit des Nutzers. Dieser Wert sollte nicht kleiner als 1 und nicht größer einer von Ihnen definierten Konstanten sein.

Prüfen Sie die Klasse `NutzerEingabe` über eine eigene Klasse `NutzerEingabeTest` mit einer `main`-Methode, in der Sie die Tests ausführen. IntelliJ bietet die Möglichkeit mit einem Rechtsklick auf `NutzerEingabeTest` unter Auswahl von `'Run NutzerEingabeTest.main()'` zu starten.

Die Klasse `Quadrat`:

Erstellen Sie im Package `de.hsos.prog3.ab02.ui` die Klasse `Quadrat`, die von der Klasse `SpielfeldDarstellung` genutzt wird, um für jede Zelle den Rahmen und bei Bedarf die Füllung eines Quadrates auf dem Interaktionsbrett zu zeichnen. Diese Klasse hat die Instanzvariable:

- `int x`: Positionswert für x auf dem Interaktionsbrett
- `int y`: Positionswert für y auf dem Interaktionsbrett
- `int seitenlaenge`: Ausdehnung des Quadrats in x und in y

Die Instanzvariablen sollen immer einen sinnvollen Wert erhalten, was über den Konstruktor der Klasse in Kombination mit den entsprechenden Setter-Methoden sicherzustellen ist.

Des Weiteren bietet die Klasse folgende Methoden an:

- `darstellenRahmen(ib:Interaktionsbrett)`: stellt ausschließlich den Rahmen des Quadrats auf dem Interaktionsbrett dar. Verwendet wird die vorhandene Methode `neuesRechteck(int x, int y, int breite, int hoehe)` des Interaktionsbretts.
- `darstellenFuellung(ib:Interaktionsbrett)`: stellt neben dem Rahmen auch eine Füllung des Quadrats auf dem Interaktionsbrett dar. Die Füllung kann bspw. so realisiert werden, dass innerhalb des Quadrats mehrere Linien über die Methode `neueLinie(int xq, int y1, int x2, int y2)` des Interaktionsbretts gezeichnet werden.
- Testen Sie die Klasse `Quadrat` über eine eigene Klasse mit einer `main`-Methode.

Die Klasse SpielfeldDarstellung

Erstellen Sie im Package `de.hsos.prog3.ab02.ui` die Klasse `SpielfeldDarstellung`. Diese Klasse hat genau eine Instanzvariable namens `ib` vom Typ `Interaktionsbrett`. Fügen Sie eine ganzzahlige Konstante für die Seitenlängen des Darstellungsbereichs auf dem Interaktionsbrett ein. Ist dieser Wert bspw. 300 und soll das Spielfeld 10*10 Zellen betragen, dann wird jede Zelle als `Quadrat`-Objekt mit einer Ausdehnung von 30*30 angezeigt. Fügen Sie zudem eine ganzzahlige Konstante `Margin` ein, um einen Abstand zwischen dem Spielfeld und dem Zeichenbereich des Interaktionsbretts einzuführen. Diese konstanten Werte sind beim Zeichnen der `Quadrat`-Objekte zu berücksichtigen.

Die Klasse bietet folgende Methoden an:

- `spielfeldDarstellen(boolean[][] spielfeld)`: erzeugt für jede Zelle ein `Quadrat`-Objekt unter Berücksichtigung der oben beschriebenen konstanten Werte und ruft dort die Methode `darstellenRahmen(...)` auf. Ist die Zelle bewohnt, wird zudem die Methode `darstellenFuellung(...)` aufgerufen. Ziel ist es, das Spielfeld als Raster aus Quadraten zu realisieren und den Zustand der Zellen über die Füllung zu visualisieren.
- `abwischen()`: ruft die Methode `abwischen` des Interaktionsbretts auf.

Testen Sie die Klasse `SpielfeldDarstellung` über eine eigene Klasse mit einer `main`-Methode.

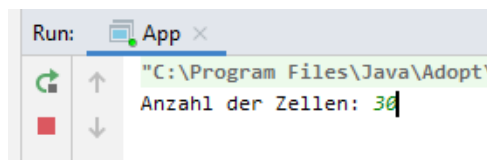
Aufgabe 2.3. Implementierung der Steuerung (4 Punkte)

Die Klasse `Steuerung` übernimmt die Interaktion mit dem Nutzer und verwendet hierzu die Klasse `NutzerEingabe`. Zudem koordiniert sie die Berechnung der Anfangs- und Folge-Generation/en durch den `Simulator` und lässt diese letztendlich über die `SpielfeldDarstellung` visualisieren. Entsprechend hat die Klasse Instanzvariablen dieser drei Klassen. Die Methode `initialisierung()` erzeugt Objekte von `Interaktionsbrett` bzw. `EinUndAusgabe` und übergibt diese an die Konstruktoren der `NutzerEingabe` bzw. `SpielfeldDarstellung`. In dieser Methode meldet sich das `Steuerung`-Objekt zudem beim `Simulation`-Objekt an, um über Änderungen informiert zu werden. Dazu implementiert die Klasse das Interface `BeiAenderung`. Die Methode `aktualisieren()` wird durch den `Simulator` aufgerufen, sobald eine Anfangs- oder Folgegeneration vorliegt. In der `aktualisieren`-Methode ruft das `Steuerung`-Objekt die Methode `spielfeldDarstellen(boolean [][])` an der entsprechenden Instanzvariable vom Typ `SpielfeldDarstellung` auf.

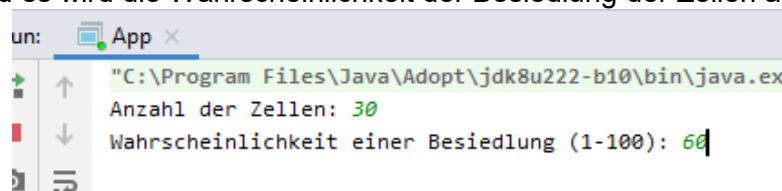
Nach der Eingabe der initialen Werte nach dem Start des Spiels durch Nutzer, wie die Größe des Spielfelds und die Wahrscheinlichkeit zur Bevölkering, hat der Nutzer anschließend die Möglichkeit zu definieren, wie viele Folgegenerationen direkt hintereinander berechnet und visualisiert werden können.

Beispielhafter Ablauf des Spiels:

1. Nach dem Start der Anwendung ruft das `Steuerung`-Objekt die Methode `anzahlZellenDesSpielfelds()` der Instanzvariablen mit dem Typ `NutzerEingabe` auf und der Nutzer wird aufgefordert, die Anzahl der Zellen des Spielfeldes anzugeben (Eingabe des Wertes 30 generiert ein Spielfeld der Größe 30*30):

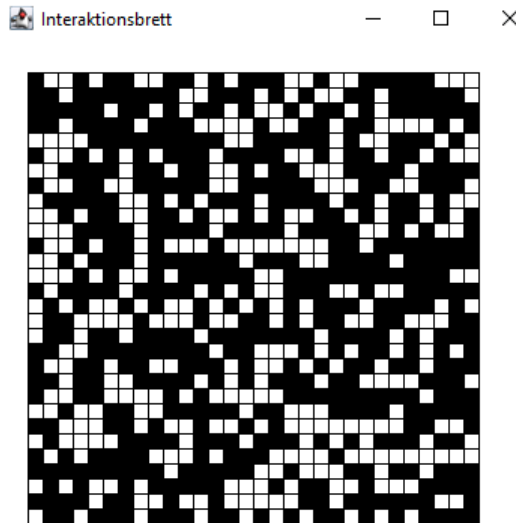


2. Anschließend verwendet das `Steuerung`-Objekt die Methode `wahrscheinlichkeitDerBesiedlung()` des verwalteten `NutzerEingabe`-Objektes und es wird die Wahrscheinlichkeit der Besiedlung der Zellen abgefragt:

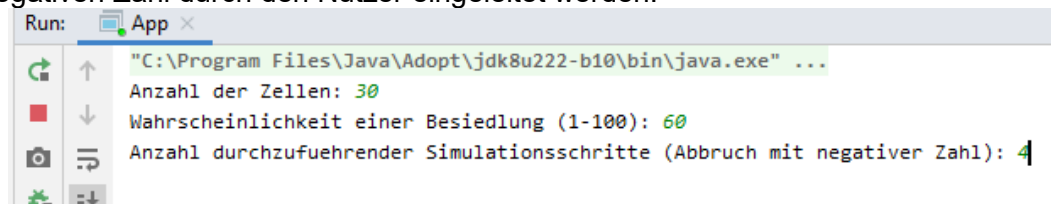


3. Das `Steuerung`-Objekt ruft dann die Methode `berechneAnfangsGeneration(anzahlZellen, wahrscheinlichkeitDerBesiedlung)` der Instanzvariable vom Typ `Simulation` auf
4. Die erste Generation wird durch den `Simulator` berechnet, das berechnete Spielfeld als zweidimensionales `boolean`-Array an die `Steuerung` durch Aufruf der `aktualisiere`-Methode übergeben und von diesem über die Methode

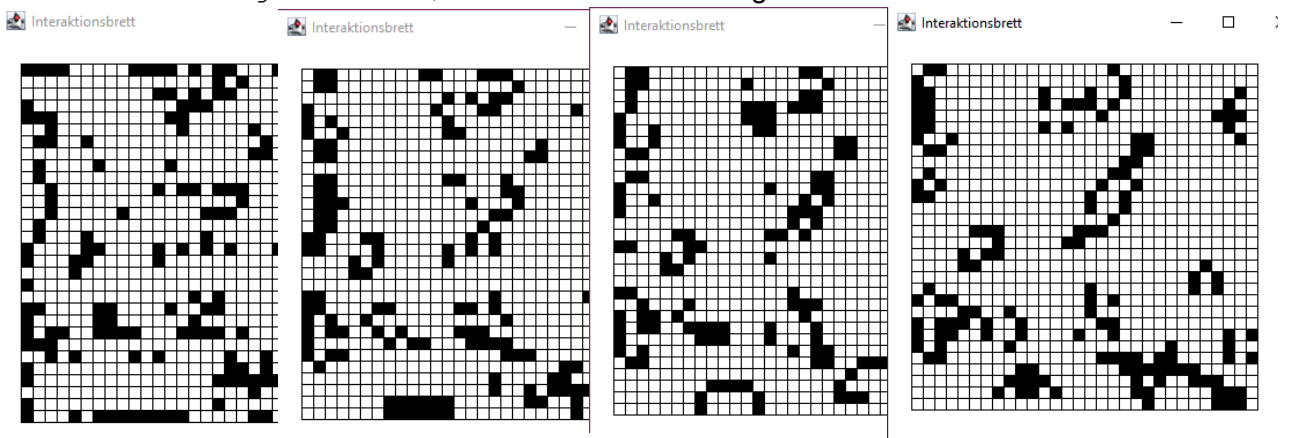
spielfeldDarstellen(...) der Instanzvariable vom Typ
SpielfeldDarstellung visualisiert:



5. Der Nutzer hat nun die Möglichkeit, die Anzahl hintereinander durchzuführender Simulationsschritte zu definieren. Der Abbruch des Spiels kann mit Eingabe einer negativen Zahl durch den Nutzer eingeleitet werden:

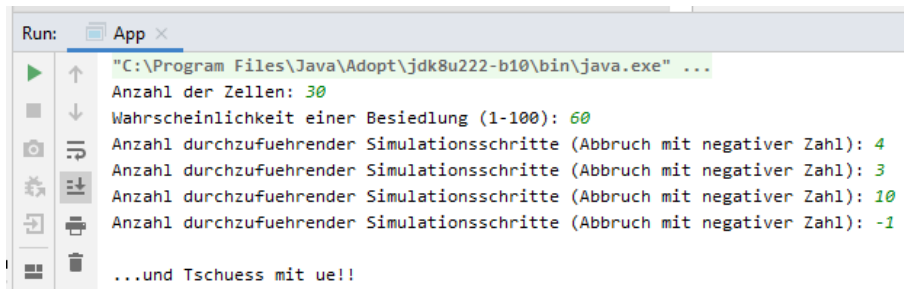


6. Anschließend werden vier Folgegenerationen hintereinander berechnet und visualisiert, ohne dass der Nutzer einschreiten kann.
Zur Information: die Geschwindigkeit der Darstellung kann bspw. über `Thread.sleep(150)` in der Methode `berechneGeneration(int berechnungsschritte)` der Klasse `Simulator` gesteuert werden.



Wenn keine Änderungen zwischen zwei Folgegenerationen auftreten, beendet das Programm die Berechnung neuer Folgegenerationen automatisch und zeigt die letzte vorliegende Generation an.

7. Mit Eingabe einer negativen Zahl, kann das Programm durch den Nutzer beendet werden:



```
Run: App x
"C:\Program Files\Java\Adopt\jdk8u222-b10\bin\java.exe" ...
Anzahl der Zellen: 30
Wahrscheinlichkeit einer Besiedlung (1-100): 60
Anzahl durchzufuehrender Simulationsschritte (Abbruch mit negativer Zahl): 4
Anzahl durchzufuehrender Simulationsschritte (Abbruch mit negativer Zahl): 3
Anzahl durchzufuehrender Simulationsschritte (Abbruch mit negativer Zahl): 10
Anzahl durchzufuehrender Simulationsschritte (Abbruch mit negativer Zahl): -1
...und Tschuess mit ue!!
```

Viel Erfolg!!