Programmierkurs Java

Dr.-Ing. Dietrich Boles

Aufgaben zu UE 9 – Klassen und Objekte I

(Stand 22.12.2015)

Aufgabe 1:

Bei dieser Aufgabe geht es darum, ein Java-Programm zu schreiben, das es einem Spieler ermöglicht, eine Variante des Spiels *Memory*, das Sie sicher noch aus Ihrer Kindheit kennen, zu spielen.

Bei dieser Variante besteht das Spiel aus n Karten, die in quadratischer Form verdeckt auf den Tisch gelegt werden, d.h. es gibt Wurzel(n) Reihen und Wurzel(n) Spalten. Wurzel (n) sei die Feldgröße. Die Feldgröße sei eine gerade natürliche Zahl größer gleich 2.

Die Karten repräsentieren Zahlen zwischen 1 und n/2. Jede dieser Zahlen kommt genau zweimal vor; wo wird per Zufall bestimmt. Die Karten sind verdeckt, so dass der Spieler anfangs nicht weiß, wo welche Zahlen versteckt sind.

In jedem Spielzug wählt der Spieler hintereinander zwei unterschiedliche Karten aus, wobei die ausgewählte Karte jeweils aufgedeckt wird. Ziel eines Spielzugs ist, dass die beiden ausgewählten Karten dieselbe Zahl repräsentieren. Ist dies der Fall, bleiben sie aufgedeckt; ist dies nicht der Fall werden sie nach dem Spielzug wieder verdeckt.

Ziel des Spiels ist es, in möglichst kurzer Zeit und mit möglichst wenigen Spielzügen alle Karten aufzudecken.

Aufgabe: Schreiben Sie ein Java-Programm, das zunächst die Feldgröße einliest und anschließend einem Spieler erlaubt, die obige Memory-Variante zu spielen. Repräsentieren Sie Spielzüge durch Objekte einer geeignet definierten Klasse.

Aufgabe 2:

Schreiben Sie ein Java-Programm, das zunächst die Anzahl der Schüler einer Schulklasse und die Anzahl an Schulfächern sowie die Namen der Fächer einliest. Anschließend sollen dann die Namen eines jeden Schülers und seine entsprechenden Noten pro Fach eingelesen werden. Nach Eingabe aller Daten soll es möglich sein, ein Fach einzugeben. Zu diesem Fach soll dann zum einen die Durchschnittsnote der Klasse bestimmt und ausgegeben wird. Zum anderen sollen die Namen der besten Schüler in diesem Fach bestimmt und ausgegeben werden.

Hinweise:

- Definieren Sie zur Repräsentation der Daten geeignete Klassen.

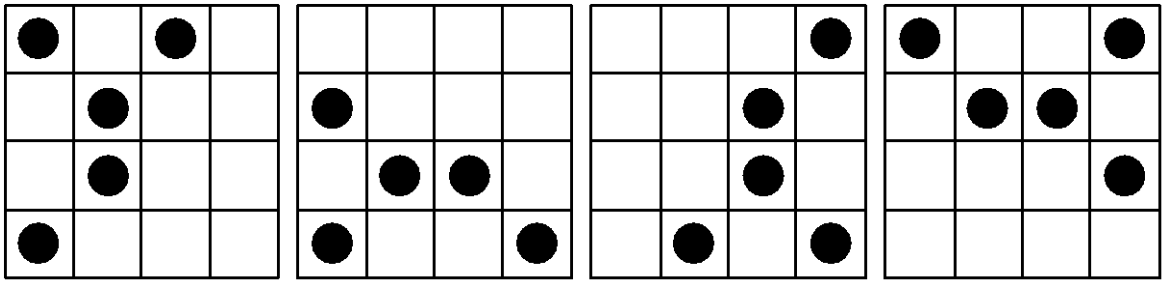
- Die Reihenfolge der einzulesenden Schulfächer muss nicht bei jedem Schüler dieselbe sein! Beispiel:

```
Schueler: Paul              Schueler: Maria
Fach: Mathematik            Fach: Deutsch
Note: 4                     Note: 3
Fach: Englisch              Fach: Englisch
Note: 3                     Note: 1
...                         ...
```

Aufgabe 3:

Puan-Puan ist ein Spiel, das von zwei Personen auf einem Spielbrett von n mal n Feldern (n = 2, 3, 4, ..., 9) gespielt wird. Als Spielfiguren stehen n mal n gleichfarbige Steine zur Verfügung. Die Spieler ziehen abwechselnd. Bei einem Zug wird entweder ein Stein gesetzt oder ein bereits gesetzter Stein weggenommen.

Wer eine Steinstellung erzeugt, die schon einmal im Verlauf des aktuellen Spiels aufgetreten ist oder die bis auf eine Drehung des gesamten Spielfeldes um 90, 180 oder 270 Grad mit einer bereits aufgetretenen Stellung übereinstimmt, verliert das Spiel. Das Spiel endet, wenn einer der beiden Spieler verliert bzw. nach 100 Zügen. In letzterem Falle endet das Spiel mit einem Unentschieden.



Vier (bis auf Drehung) gleiche Stellungen („Puan-Gleichheit“)

Aufgabe: Schreiben Sie ein Java-Programm, das zuerst die Größe des Spielbrettes einliest und dann abwechselnd die Spielzüge beider Spieler entgegennimmt, bis das Spiel endet. Repräsentieren Sie Spielzüge durch Objekte einer geeignet definierten Klasse.

Aufgabe 4:

Das „Game-of-Life“ wird auf einem schachbrettartigen Feld gespielt, das eine „Bevölkerung“ von „toten“ und „lebenden“ Zellen darstellt. Jede Zelle kann „überleben“, „sterben“ oder „geboren“ werden. Die schrittweise Entwicklung von einem Stellungsbild zum nächsten erfolgt gemäß einiger Regeln, die berücksichtigen, wie viele lebende Nachbarzellen eine Zelle hat. Eine Zelle x, die nicht am Spielfeldrand liegt, hat 8 Nachbarzellen, Zellen am Spielfeldrand entsprechend weniger.

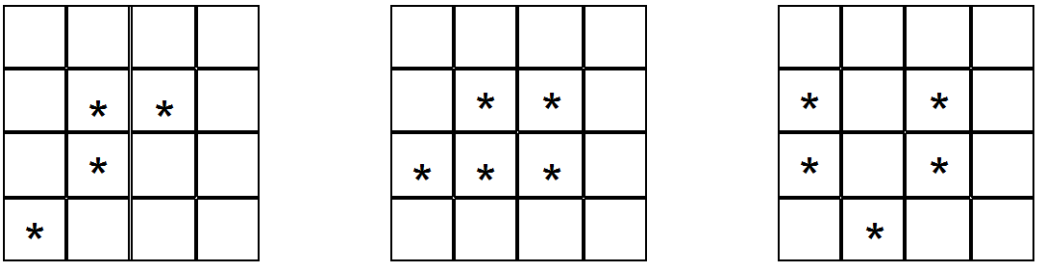
Die Regeln, nach denen sich die Population von einer Stellung zur nächsten entwickelt, sind:

- Für eine Zelle x, die gerade tot ist, gilt: Wenn x genau 3 lebende Nachbarzellen hat, wird x neu geboren; sonst bleibt x tot.

- Für eine Zelle x, die gerade lebendig ist, gilt: Wenn x weniger als 2 lebende Nachbarn hat, stirbt x an Vereinsamung; wenn x 2 oder 3 lebende Nachbarzellen hat, bleibt x in der nächsten Stellung lebendig. In allen anderen Fällen stirbt x an Überbevölkerung.

Alle Veränderungen gemäß dieser Regeln geschehen gleichzeitig. Die Simulation beginnt mit einer bestimmten eingelesenen Verteilung von lebenden und toten Zellen. Sie endet nach einer bestimmten eingelesenen Anzahl von Entwicklungsschritten, jedoch früher, wenn sich die Population nicht mehr ändert. Das Spielfeld sollte aus mindestens 15x15 Feldern bestehen.

Beispiel:



Aufgabe: Implementieren Sie das „Game-of-Life“.

Aufgabe 5:

In dieser Aufgabe geht es um die Implementierung des bekannten Spiels *Reversi*.

Spieler

Reversi ist ein Spiel für zwei Personen.

Spielbrett

Reversi wird auf einem Spielbrett gespielt, das sich aus 8*8 einzelnen Feldern zusammensetzt. Es hat also dieselben Ausmaße wie ein Schachbrett, nur das die Felder nicht abwechselnd schwarz und weiß sind, sondern eine einheitliche Farbe haben. Es ist zweckmäßig, das Brett wie beim Schach mit Koordinaten zu versehen, um Spielzüge eindeutig angeben zu können.

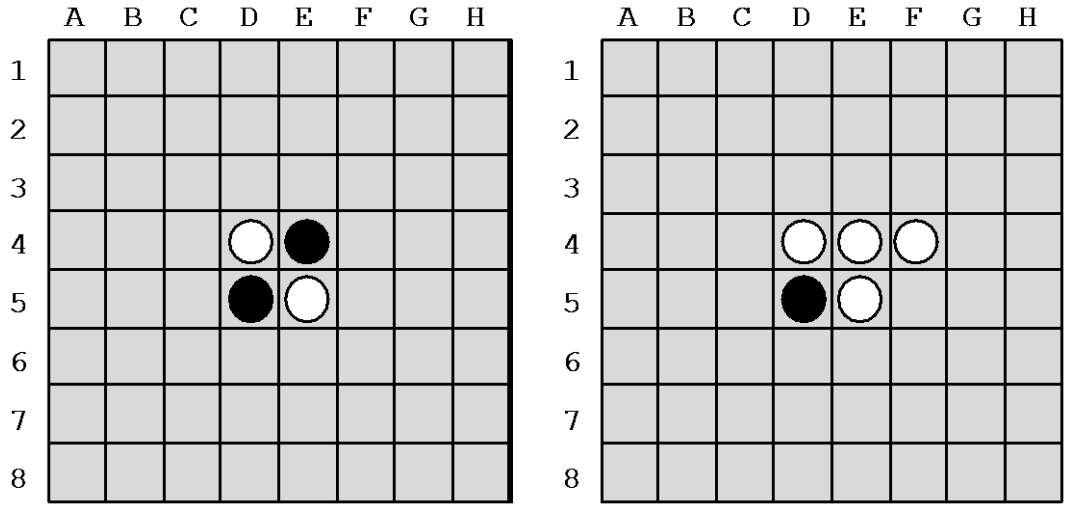
Spielfiguren

Als Spielfiguren werden runde Plättchen (Steine) verwendet. Diese haben je eine schwarze und eine weiße Seite. Es gibt insgesamt 64 Plättchen, also genauso viele, wie das Spielbrett Felder hat. Die Spielplättchen werden im Laufe des Spiels umgedreht (daher der Name Reversi), so dass schwarze zu weißen Steinen werden können und umgekehrt. Im Folgenden ist immer von schwarzen und weißen Steinen die Rede, was sich jeweils auf die Oberseite der Steine - also die sichtbare Farbe - bezieht.

Spielverlauf und Ziel des Spiels

Der Spieler, der das Spiel beginnt, wird im folgenden Spieler A genannt. Der andere ist dementsprechend Spieler B. Spieler A bekommt die Farbe „Weiß“, Spieler B die Farbe „Schwarz“ zugeteilt. Zu Anfang des Spiels befinden sich vier Steine auf dem

Spielbrett, zwei mit ihrer weißen (D4 und E5), zwei mit ihrer schwarzen Seite (E4 und D5) nach oben (siehe Abbildung links). Die restlichen 60 Steine liegen in einem Sammelpool neben dem Spielbrett.



Spieler A führt dann seinen ersten Spielzug aus: Er nimmt einen Stein aus dem Sammelpool und legt ihn auf ein leeres Feld (selbstverständlich mit seiner Farbe „Weiß“ nach oben). Das leere Feld muss dabei an ein belegtes Feld horizontal, vertikal oder diagonal angrenzen. Außerdem muss der Spielzug zum „Schlagen“ von mindestens einem gegnerischen Stein führen. „Schlagen“ bedeutet für Spieler A: Drehe alle schwarzen Steine um, die sich horizontal, vertikal oder diagonal zwischen bereits gesetzten weißen Steinen und dem neu gesetzten (ebenfalls weißen) Stein befinden. In der Ausgangsituation kann sich Spieler A also eines der Felder E3, F4, D6 und C5 aussuchen. Setzt er den Stein bspw. auf Feld F4, muss er den schwarzen Stein auf Feld E4 umdrehen (siehe Abbildung rechts).

Damit ist der erste Spielzug von Spieler A beendet und Spieler B kommt zum Zug. Dieser führt nun eine identische Aktion durch, nur dass er jetzt natürlich weiße Steine schlagen muss.

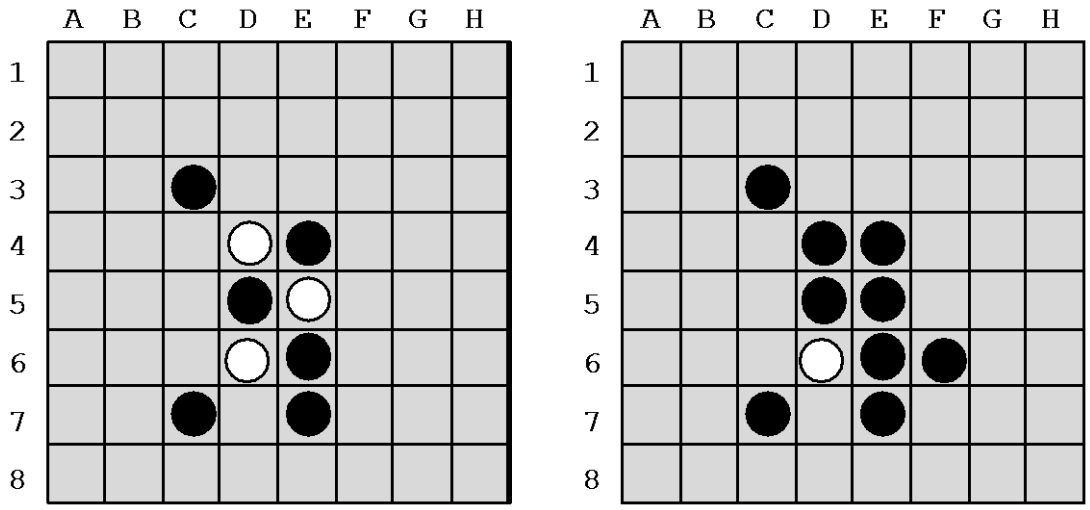
Die beiden Spieler führen abwechselnd ihre Spielzüge durch. Ist es einem Spieler nicht möglich, ein leeres Feld mit einem Stein zu besetzen, so muss er passen und der andere Spieler ist wieder am Zug.

Das Spiel ist beendet, wenn kein Spieler mehr einen Stein setzen kann. Dies ist in der Regel der Fall, wenn alle Felder besetzt sind, kann aber auch schon vorher passieren. Die schwarzen und weißen Steine auf dem Spielbrett werden gezählt. Sieger des Spiels ist der Spieler, der die größere Anzahl an Steinen auf dem Spielbrett hat.

Achtung:

- Ein Stein, der einmal auf dem Spielbrett liegt, wird nie mehr vom Brett genommen oder verschoben. Er wird höchstens umgedreht.
- Jeder Spielzug muss vollständig ausgeführt werden, d.h. alle Steine, die aufgrund eines neu gesetzten Steines geschlagen werden können, müssen auch umgedreht werden. Die Spieler dürfen sich nicht aussuchen, welche Steine sie umdrehen und welche nicht.

- Solange ein Spieler gegnerische Steine schlagen kann, muss er ziehen. Er darf nicht freiwillig passen.
- Der entscheidende Stein beim Umdrehen ist der neu gesetzte Stein. Zum Beispiel werden weiße Steine nicht umgedreht, die zwischen zwei schwarzen Steinen liegen, die beide schon vorher auf dem Spielbrett lagen. Es findet also keine transitive Fortsetzung beim Umdrehen von Steinen statt. Wird bspw. bei der Ausgangsposition in der Abbildung unten links ein schwarzer Stein auf Feld F6 gesetzt, dann werden die beiden weißen Steine auf den Feldern D4 und E5 umgedreht, und es ergibt sich die Situation in der Abbildung unten rechts. Obwohl durch die Umdreh-Aktion nun auch der weiße Stein auf Feld D6 zwischen zwei schwarzen Steinen (E5 und C7) liegt, wird dieser nicht umgedreht, weil sich die beiden schwarzen Steine schon vorher auf dem Spielbrett befanden.



Aufgabe: Implementieren Sie das Spiel Reversi, so dass es 2 Menschen gegeneinander an einer Konsole spielen können. Definieren und benutzen Sie zur Repräsentation von Spielzügen folgende Klasse:

```
class ReversiSpielzug {
     int reihe;   // Reihe
     char spalte; // Spalte
}
```

Aufgabe 6:

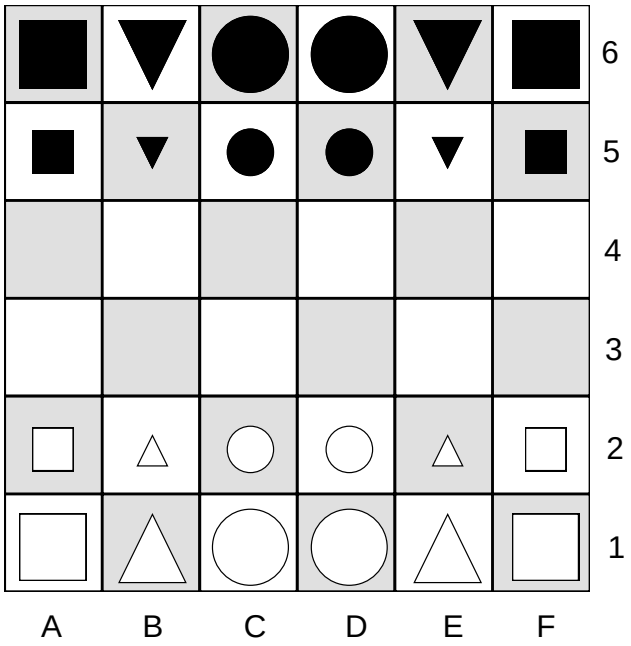
In dieser Aufgabe geht es um die Implementierung des Spiels *Opala*.

Spielbrett

Gespielt wird auf einem Schachbrett mit allerdings nur 6x6 Feldern.

Spielfiguren

Spielfiguren sind geometrische Gebilde. Es gibt große und kleine Quadrate, große und kleine Kreise sowie große und kleine Dreiecke. In der Abbildung ist die Anfangsaufstellung zu sehen.



Spieler

OPALA ist ein Spiel für zwei Spieler. Spieler A (Weiß) spielt mit weißen Spielfiguren von unten nach oben. Spieler B (Schwarz) spielt mit schwarzen Spielfiguren von oben nach unten.

Spielablauf

Es wird immer abwechselnd gezogen, wobei Spieler Weiß beginnt. Es besteht Zugzwang.

Spielzüge

Für die einzelnen Spielfiguren gelten folgende Zugregeln:

- Große Quadrate dürfen wie Türme beim Schach gezogen werden, d.h. beliebig (>0) viele Felder horizontal oder vertikal.
- Kleine Quadrate dürfen wie große Quadrate gezogen werden, allerdings nur ein einzelnes Feld.
- Große Dreiecke dürfen wie Läufer beim Schach gezogen werden, d.h. beliebig (>0) viele Felder in einer Diagonalen.
- Kleine Dreiecke dürfen wie große Dreiecke gezogen werden, allerdings nur ein einzelnes Feld.
- Große Kreise dürfen wie Damen beim Schach gezogen werden, d.h. beliebig (>0) viele Felder in einer Horizontalen, Vertikalen oder Diagonalen.
- Kleine Kreise dürfen wie große Kreise gezogen werden, allerdings nur ein einzelnes Feld

Dabei gelten grundsätzlich folgende Resultate bzw. Einschränkungen:

- Bei einem Spielzug darf nur eine einzelne eigene Spielfigur gezogen werden.
- Bei einem Spielzug dürfen keine Spielfiguren übersprungen werden.
- Es darf nicht auf ein Feld gezogen werden, auf dem bereits eine eigene Spielfigur steht.

- Wird eine Spielfigur auf ein Feld gezogen, auf dem eine Spielfigur des Gegners steht, so wird diese geschlagen, d.h. vom Spielbrett entfernt.

Ziel des Spiels

Ziel des Spiels ist es, mit einer eigenen Spielfigur die gegenüber liegende Grundlinie zu erreichen, und zwar ohne dass diese im nächsten Zug wieder geschlagen werden könnte.

Ende des Spiels und Sieger

Ein Spiel ist beendet:

- wenn ein Spieler X mit einer eigenen Spielfigur die gegenüber liegende Grundlinie erreicht hat und diese Spielfigur im nächsten Zug nicht direkt geschlagen werden kann. In diesem Fall hat Spieler X unmittelbar gewonnen.
- wenn ein Spieler X mit einer eigenen Spielfigur die gegenüber liegende Grundlinie erreicht hat und diese Spielfigur zwar im nächsten Zug direkt geschlagen werden kann, sein Gegner dies jedoch nicht tut. In diesem Fall hat Spieler X gewonnen, nachdem der Gegner seinen Zug ausgeführt hat.
- wenn ein Spieler, wenn er am Zug ist, keinen legalen Spielzug mehr ausführen kann. In diesem Fall hat sein Gegner gewonnen.
- sobald ein Spieler keine eigenen Spielfiguren mehr besitzt. In diesem Fall hat der Gegner gewonnen.

Aufgabe: Implementieren Sie das Spiel Opala, so dass es 2 Menschen gegeneinander an einer Konsole spielen können. Definieren und benutzen Sie zur Repräsentation von Spielzügen folgende Klasse:

```
class OpalaSpielzug {
     int reihe;   // Reihe
     char spalte; // Spalte
}
```

Aufgabe 7:

In dieser Aufgabe geht es um die Implementierung des Spiels *Metamorphose*.

Spieler

Metamorphose ist ein Spiel für zwei Personen (Spieler A und Spieler B).

Spielbrett

Metamorphose wird auf einem Spielbrett gespielt, das sich aus 7*7 einzelnen Feldern zusammensetzt. Bestimmte Felder sind durch Mauern besetzt (siehe Abbildung). Es ist zweckmäßig, das Brett wie beim Schach mit Koordinaten zu versehen, um Spielzüge eindeutig angeben zu können.

Spielfiguren

Als Spielfiguren werden Quadrate und Kreise verwendet. Es gibt weiße Spielfiguren, die Spieler A gehören, und schwarze Spielfiguren, die Spieler B gehören. Anfangs besitzt jeder Spieler vier Quadrate und drei Kreise.

Spielzug

In jedem Spielzug verschiebt ein Spieler eine seiner Spielfiguren um ein oder mehrere Felder auf dem Spielbrett. Dabei gilt: Quadrate können horizontal und vertikal verschoben werden. Kreise können diagonal verschoben werden. Nachdem eine Figur verschoben wurde, wechselt sie ihren Typ (aus einem Kreis wird ein Quadrat und umgekehrt).

Spielregeln

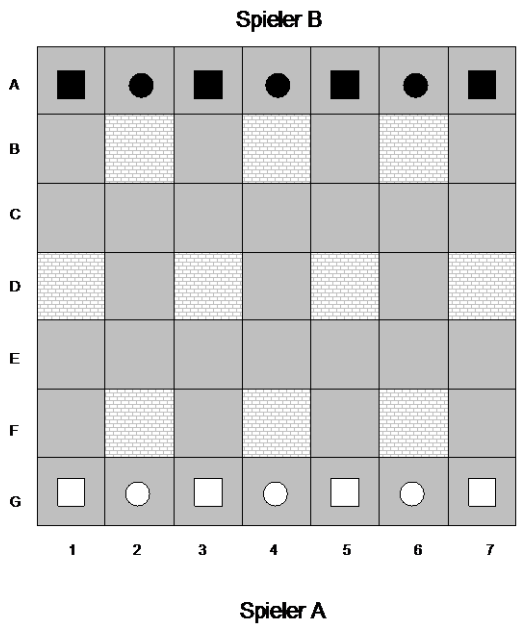
- Die Anfangsaufstellung ist in der Abbildung dargestellt.
- Spieler A (Weiß) beginnt.
- Es besteht Zugzwang. Kann ein Spieler nicht mehr ziehen kann, hat er verloren!
- Auf einem Feld darf maximal eine Figur stehen.
- Auf die durch Mauern besetzten Felder dürfen keine Figuren gezogen werden.
- Bei einem Spielzug dürfen keine Mauern und andere Figuren (der eigenen oder fremden Farbe) übersprungen werden.
- Endet ein Spielzug auf einem Feld, auf dem eine fremde Figur steht, so gilt diese als geschlagen und wird vom Spielbrett entfernt.
- Eigene Figuren dürfen nicht geschlagen werden.

Ziel des Spiels

Am Anfang des Spiels wählt jeder Spieler (geheim) eine seiner Spielfiguren aus und merkt sie sich als besondere Spielfigur, den König. Wer als erster den König des Gegners schlägt, hat gewonnen. Das Spiel endet unentschieden, wenn beide Spieler nur noch ihren König besitzen.

Aufgabe: Implementieren Sie das Spiel Metamorphose, so dass es 2 Menschen gegeneinander an einer Konsole spielen können. Definieren und benutzen Sie zur Repräsentation von Spielzügen folgende Klasse:

```
class MetamorphoseSpielzug {
     char reihe;  // Reihe
     int spalte;  // Spalte
}
```



Aufgabe 8:

Bei dieser Aufgabe sollen Sie ein Programm entwickeln, durch das ein menschlicher Spieler am Computer das Knobelspiel *Lampen* spielen kann.

Regeln:

Das Spiel besteht aus einem quadratischen Spielfeld der Größe *n* ($2 < n < 10$), das aus einzelnen Zellen besteht, die Lampen repräsentieren. Lampen können sich im Zustand *an* oder *aus* befinden. Anfangs sind alle Lampen im Zustand *aus*. In jedem Spielzug wählt der Spieler eine Zelle aus, deren Zustand umgeschaltet werden soll (von *aus* in *an* oder umgekehrt). Gleichzeitig werden aber auch die Nachbarlampen oberhalb, unterhalb, links und rechts von der entsprechenden Lampe umgeschaltet (insofern sie existieren). Ziel (und Ende) des Knobelspiels ist es, den Zustand zu erreichen, dass alle Lampen angeschaltet sind.

Aufgabe:

Schreiben Sie ein Java-Programm, mit dessen Hilfe ein menschlicher Spieler das Spiel *Lampen* spielen kann. Definieren und benutzen Sie zur Repräsentation von Spielzügen folgende Klasse:

```
class LampenSpielzug {
     int reihe;  // Reihe
     int spalte; // Spalte
}
```

Hinweise:

Der generelle Spielablauf lässt sich folgendermaßen skizzieren:

- Abfrage der Spielfeldgröße
- Initialisierung des Spielfeldes
- Ausgabe des Spielfeldes
- Solange das Spiel nicht beendet ist, tue folgendes
    - Korrekten Spielzug einlesen
    - Spielzug ausführen
    - Ausgabe des Spielfeldes

Beispiel für einen Programmablauf (Benutzereingaben stehen in Klammern (<>), Lampen im Zustand *aus* werden durch ein ‚.', Lampen im Zustand *an* durch ein ‚+' gekennzeichnet):

```
Feldgroesse (2 < n < 10): <5>

 01234
0.....
1.....
2.....
3.....
4.....

Reihe der Lampe, die umgeschaltet werden soll (0 <= r < 5): <3>
Spalte der Lampe, die umgeschaltet werden soll (0 <= r < 5): <2>
```

```
 01234
0.....
1.....
2..+..
3.+++.
4..+..

Reihe der Lampe, die umgeschaltet werden soll (0 <= r < 5): <0>
Spalte der Lampe, die umgeschaltet werden soll (0 <= r < 5): <0>

 01234
0++...
1+....
2..+..
3.+++.
4..+..

Reihe der Lampe, die umgeschaltet werden soll (0 <= r < 5): <3>
Spalte der Lampe, die umgeschaltet werden soll (0 <= r < 5): <1>

 01234
0++...
1+....
2.++..
3+..+.
4.++..

...
```

Aufgabe 9:

Bei dieser Variante des Nim-Spiel sind n Reihen mit n Streichhölzern vorhanden. Zwei Spieler nehmen abwechselnd Streichhölzer aus einer der Reihen weg. Wie viele sie nehmen, spielt keine Rolle; es muss mindestens ein Streichholz sein und es dürfen bei einem Zug nur Streichhölzer einer einzigen Reihe genommen werden. Derjenige Spieler, der den letzten Zug macht, also die letzten Streichhölzer wegnimmt, gewinnt.

Implementieren Sie diese Variante des Nim-Spiels in Java, so dass zwei menschliche Spieler gegeneinander antreten können. Definieren und benutzen Sie zur Repräsentation von Spielzügen folgende Klasse:

```
class NimSpielzug {
     int haufen; // Nummer des Haufens, von dem genommen werden soll
     int anzahl; // Anzahl der genommenen Streichhoelzer
}
```

Aufgabe 10:

Rushhour ist ein Spiel für einen einzelnen Spieler. Auf einem rechteckigen Spielbrett, das aus einzelnen Kacheln besteht, wird initial eine Menge an Autos platziert. Jedes Auto überdeckt mindestens zwei Kacheln. Ein Auto mit mehr als zwei Kacheln darf dabei keine Ecken aufweisen. Liegen die Kacheln, die ein Auto überdeckt, nebeneinander, ist es ein horizontal verschiebbares Auto. Liegen die Kacheln übereinander, ist es ein vertikal verschiebbares Auto. Ein Auto wird als Hauptauto gekennzeichnet. Ziel des Spiels ist es nun, die Autos so zu verschieben, bis irgendwann das Hauptauto (im Bild bspw. das rote Auto) den rechten Rand berührt.

Horizontale Autos dürfen dabei nur nach links und rechts, vertikale Autos nur nach oben und unten verschoben werden. Ein Auto darf nicht über den Rand hinweg oder wenn ein anderes Auto im Wege steht, verschoben werden.

Entwickeln Sie ein Java-Programm, mit dem ein Benutzer Rushhour spielen kann. Die Ausgangsstellung der Autos sei dabei in einer Textdatei festgelegt, die mittels einer der readFile-Methoden der Klasse IO gelesen werden kann. Die Autoteile werden dabei durch jeweils gleiche Zeichen repräsentiert; das Hauptauto durch das Zeichen ‚*'. Die Stellung der Autos in der oberen Abbildung entspräche bspw. folgendem Dateiinhalt:

```
122 3
145 36
145**6
7778 6
  98aa
bb9cc
```

Der Benutzer soll nun solange Autos verschieben können (durch Angabe von Auto und Richtung) bis das Hauptauto den rechten Rand erreicht.

Überlegen Sie sich eine geeignete Datenstruktur zur Repräsentation des Spielbretts und von Spielzügen.

Im Folgenden wird ein beispielhafter Programmablauf skizziert (Benutzereingaben in <>):

```
122 3
145 36
145**6
7778 6
  98aa
bb9cc

Wahl eines Autos (Buchstabe angeben): <c>
Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <>r
```

```
122 3
145 36
145**6
7778 6
  98aa
bb9 cc

Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <q>
Wahl eines Autos (Buchstabe angeben): 8
Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <d>
122 3
145 36
145**6
777  6
  98aa
bb98cc

Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <q>
Wahl eines Autos (Buchstabe angeben): 7
Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <r>
122 3
145 36
145**6
 777 6
  98aa
bb98cc

Richtung (l=links, r=rechts, u=hoch, d=runter, q=Autowechsel): <r>
122 3
145 36
145**6
  7776
  98aa
bb98cc

...
```

Aufgabe 11:

In dieser Aufgabe geht es um die Implementierung des Spiels *TicTacToe*.

Spielregeln:

Auf einem 3×3 Felder großen Spielfeld machen zwei Spieler abwechselnd ihre Zeichen (ein Spieler Kreuze, der andere Kreise). Der Spieler, der als erstes drei seiner Zeichen in eine Reihe, Spalte oder eine der beiden Hauptdiagonalen setzen kann, gewinnt. Wenn dies keinem der beiden Spieler gelingt, kommt es zu einem Unentschieden. Siehe auch http://de.wikipedia.org/wiki/Tic_Tac_Toe

Aufgabe:

Implementieren Sie das Spiel TicTacToe, so dass es 2 Menschen gegeneinander an einer Konsole spielen können. Definieren und benutzen Sie zur Repräsentation von Spielzügen folgende Klasse:

```
class TicTacToeSpielzug {
     int zeile;  // Zeile
     int spalte; // Spalte
}
```

Im Folgenden wird ein beispielhafter Programmablauf skizziert (Benutzereingaben in <>):

```
   0 1 2
 -------+
 0| | | |
 -------+
 1| | | |
 -------+
 2| | | |
 -------+
 Spieler A ist am Zug!
 Zeile (0-2): <0>
 Spalte (0-2): <0>
   0 1 2
 -------+
 0|X| | |
 -------+
 1| | | |
 -------+
 2| | | |
 -------+
 Spieler B ist am Zug!
 Zeile (0-2): <1>
 Spalte (0-2): <0>
   0 1 2
 -------+
 0|X| | |
 -------+
 1|O| | |
 -------+
 2| | | |
 -------+
 Spieler A ist am Zug!
 Zeile (0-2): <1>
 Spalte (0-2): <1>
   0 1 2
 -------+
 0|X| | |
 -------+
 1|O|X| |
 -------+
 2| | | |
 -------+

 ...
```

Aufgabe 12:

Das so genannte Normalgewicht berechnet sich nach der Formel "Körpergröße (in cm) minus 100". Das Idealgewicht beträgt bei Männern 90% und bei Frauen 85% des Normalgewichts.

Schreiben Sie ein Java-Programm, welches nach Eingabe von Größe, Gewicht und Geschlecht ausgibt, ob ein Mensch zu dick (über Normalgewicht) oder zu dünn (unter Idealgewicht) ist, oder ob er/sie zwischen Ideal- und Normalgewicht liegt.

Definieren Sie zur Repräsentation eines Menschen eine adäquate Klasse und nutzen Sie diese (Mensch: Geschlecht, Größe, Gewicht).

Im Folgenden wird ein beispielhafter Programmablauf skizziert (Benutzereingaben in <>):

```
Groesse (cm) eingeben: <181>
Gewicht (kg) eingeben: <73>
Maennlich (m/w): <m>
Idealer Gewichtsbereich!
Weitere Ueberpruefung (j/n)?<j>
Groesse (cm) eingeben: <172>
Gewicht (kg) eingeben: <95>
Maennlich (m/w): <w>
Alter Fettsack!
Weitere Ueberpruefung (j/n)? ...
```

Aufgabe 13:

In der Schule haben sie Brüche kennengelernt. Ein Bruch besteht aus einem Zähler und einem Nenner (beides int-Werte). Brüche kann man addieren, multiplizieren, vergleichen, …

Definieren Sie zunächst eine Klasse `Bruch`. Implementieren Sie dann Funktionen zum Addieren, Multiplizieren, Vergleichen usw. von Brüchen. Denken Sie ans Kürzen.

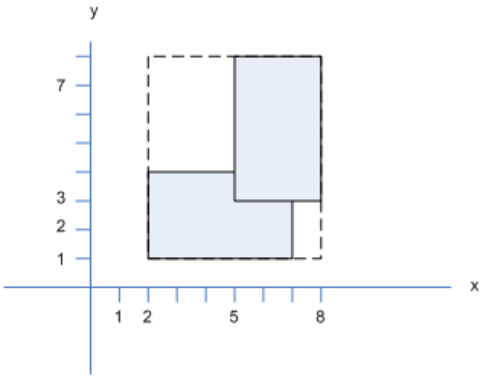
```
static void init(Bruch b, int zaehler, int nenner)
static void addTo(Bruch b1, Bruch b2) // b1 = b1 + b2
static void multTo(Bruch b1, Bruch b2) // b1 = b1 * b2
static boolean gleich(Bruch b1, Bruch b2) // b1 == b2
static String getString(Bruch b) // z.B. 3/4 oder 7/8
...
```

Aufgabe 14:

Gegeben sei ein kartesisches Koordinatensystem. Rechtecke lassen sich hierin eindeutig beschreiben durch die Koordinaten des Punktes ihrer linken unteren Ecke (beides int-Werte) sowie ihre Breite und Höhe (beides positive int-Werte).

- Definieren Sie eine Klasse `Rechteck`, durch die entsprechende Rechtecke repräsentiert werden können.
- Implementieren Sie dann eine Funktion, die zwei Rechtecke als Parameter übergeben bekommt. Die Funktion soll das flächenmäßig kleinstmögliche Rechteck berechnen, erzeugen und als Funktionswert liefern, das beide übergebenen Rechtecke umschließt.
- Schreiben Sie ein kleines Testprogramm, in dem der Benutzer entsprechende Werte zweier Rechtecke eingibt, die Funktion aufgerufen wird und die Werte des gelieferten Rechteckes ausgegeben werden. Sie können davon ausgehen, dass der Benutzer korrekte Werte eingibt (also nur positive Werte für Breite und Höhe).

Die folgende Abbildung skizziert das Szenario.



Ein Beispielablauf für das Testprogramm mit den Rechtecken in der Abbildung (Benutzereingaben in <>):

```
Rechteck eingeben:
x: <2>
y: <1>
width: <5>
height: <3>
Rechteck eingeben:
x: <5>
y: <3>
width: <3>
height: <5>
Rechteck ausgeben:
x = 2
y = 1
width = 6
height = 7
```

Aufgabe 15:

Das SOS-Spiel ist ein Spiel für 2 Personen. Das Spielbrett besteht aus N (hier N = 20, aber prinzipiell beliebig > 1) nebeneinander angeordneten (zunächst leeren) Feldern. Die Spieler A (Anziehender) und B (Nachziehender) füllen abwechselnd die Felder in beliebiger Reihenfolge aus; es sind nur die Einträge S oder O erlaubt. Der Spieler, der als erster eine Teilfolge ...SOS... erzeugt, ist Sieger. Sind alle Felder belegt, ohne dass es die Teilfolge SOS gibt, endet das Spiel mit einem Unentschieden. Es besteht Zugzwang. Nur gültige Spielzüge werden akzeptiert.

Aufgabe: Implementieren Sie (mit imperativen Programmierkonzepten) das SOS-Spiel, so dass es zwei menschliche Spieler auf einer Konsole gegeneinander spielen können. Ein gültiger Spielzug besteht dabei aus der Angabe des Index (zwischen 0 und N-1) und einem Buchstaben (S oder O).

Beispiel für einen möglichen Spielablauf (Benutzereingaben in <>):

```
....................
Spieler 1 ist am Zug!
Index eingeben: <1>
S oder O eingeben: <S>
.S..................
Spieler 2 ist am Zug!
Index eingeben: <7>
S oder O eingeben: <O>
.S.....O............
```

```
Spieler 1 ist am Zug!
Index eingeben: <3>
S oder O eingeben: <T>
Ungueltiger Buchstabe! S oder O eingeben: <S>
.S.S...O.....
Spieler 2 ist am Zug!
Index eingeben: <2>
S oder O eingeben: <O>
.SOS...O.....
Spieler 2 hat gewonnen!
```