TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Angewandte Informatik / Kooperative Systeme
Praktikum Grundlagen der Programmierung
Prof. Dr. A. Brüggemann-Klein, R. Palenta, J. Kranz, Dr. A. Reuss

WS 15/16
Übungsblatt 5
13.11.2015

Abgabe: 23.11.2015 (vor 5:00 Uhr)

Hinweis: Implementieren Sie alle Hausaufgaben in einem Package namens **blatt05**.

Aufgabe 5.1 (P) Jasmin I

Betrachten Sie die Klasse `Haus`. Geben Sie zu dieser den Callstack entsprechend der JASM.IN-Notation aus der Vorlesung an, wobei der Nutzer als Hausbreite 12 und als Innenraumbreite 4 angegeben hat und

a) der oberste Aufruf auf dem Callstack die Methode `dach()` in Zeile 16 ist und die beiden for-Schleifen in der Methode `dach()` gerade zum letzten Mal durchlaufen werden.

b) der oberste Aufruf auf dem Callstack die Methode `dach()` in Zeile 23 ist, die for-Schleife in Zeile 15 maximal oft durchlaufen wurde und die for-Schleife in Zeile 13 zum letzten Mal durchlaufen wird.

Den Wert für die Variable `scanner` sollten Sie leer lassen.

```
1  import java.util.Scanner;
2
3  public class Haus {
4
5      //Breite des Hauses
6      static int h;
7      //Breite des Innenraums
8      static int i;
9
10   public static void dach () {
11
12     //Iteration ueber Zeilen, Dach ist halb so hoch wie Haus breit
13     for (int i = 0; i < h/2; i++) {
14         //Iteration ueber Spalten
15         for(int j = 0; j < h; j++){
16             if(j < (h/2)-i-1 || j > h-(h/2-i))
17                  //Ausgabe Leerzeichen
18                  System.out.print(' ');
19             else
20                 // Ausgabe Dach
21                 System.out.print('*');
22         }
23         System.out.println();
24     }
25   }
26
27   public static void mauer () {
28
29     //Balkenbreite
30     int m = (h-i)/2;
31
32     //Iteration ueber Zeilen, Dochhoehe ist konstant 6
33     for(int i = 0; i < 6; i++){
34         //Iteration ueber Spalten
```

```
35          for(int j = 0; j < h; j++){
36              //Ausgabe der Boden- und Deckenzeile
37              if(i == 0 || i ==  5)
38                  System.out.print('+');
39              else{
40                  if(j < m || j > h-m-1)
41                      System.out.print('+');
42                  else
43                      System.out.print(' ');
44              }
45          }
46          System.out.println();
47      }
48
49    }
50
51    public static void main (String[] args) {
52
53        Scanner scanner = new Scanner(System.in);
54
55        boolean repeat = true;
56
57        while(repeat){
58            System.out.println("Wieviele Zeichen soll das Haus breit sein? (
                  Die Anzahl der Zeichen muss gerade sein.)");
59            h = scanner.nextInt();
60            if(h > 1 && h % 2 == 0)
61                repeat = false;
62        }
63        repeat = true;
64        while(repeat){
65            System.out.println("Wieviele Zeichen soll der Innenraum breit
                  sein? (Die Anzahl der Zeichen muss gerade sein und "
66                      + "kleiner als " + h + ")");
67            i = scanner.nextInt();
68            if(i >= 0 && i < h && i % 2 == 0)
69                repeat = false;
70        }
71
72      dach();
73      mauer();
74    }
75  }
```

Hinweis: Wenn der oberste Aufruf auf dem Callstack die Methode in Zeile X ist, bedeutet das, dass die Zeile X noch nicht ausgeführt wurde.

Aufgabe 5.2 (P) Schlangenspiel

In dieser Aufgabe sollen Sie das Schlangenspiel für zwei Spieler programmieren. Erstellen Sie dazu ein Programm namens `Schlangenspiel.java`. Die Grundidee des Spiels ist folgende:

- Das Spiel hat die Felder 0 bis 35.
- Jeder Spieler besitzt einen Spielstein.
- Beide Spielsteine starten auf Feld 0.
- Im Wechsel wird gewürfelt. Der Spielstein des entsprechenden Spielers wird um die entsprechende Augenzahl vorgerückt.
- Wer zuerst das Feld 35 erreicht oder überschreitet, gewinnt das Spiel.
- Von allen Feldern, die durch 5 teilbar sind, führt eine Leiter nach oben. Wer ein solches Feld erreicht, kommt sofort 3 Felder weiter.

- Aber Vorsicht vor den Schlangen! Erreicht man ein Schlangenfeld (jedes Feld, das durch 7 teilbar ist), rutscht man automatisch um 4 Felder zurück.
- Die Felder 0 und 35 sind weder Leiter- noch Schlangenfelder.
- Leitern und Schlangen treten in Aktion, wenn ein Spielstein dieses Feld erreicht. Es ist daher möglich, *Ketten* von Schlangen und/oder Leitern zu benutzen.
- Es dürfen zwei Spielsteine gleichzeitig auf einem Feld stehen.
- Am Ende wird der Sieger ausgegeben.

Geben Sie jeweils das Würfelergebnis und das neue Spielfeld aus.

Implementieren Sie ihre Lösung Schritt für Schritt:

a) Beschränken Sie sich zuerst auf einen Spieler und ein leeres Spielfeld. Das Spiel besteht am Anfang nur aus Würfeln bis die 35 überschritten wird.
b) Beachten Sie nun Leitern und Schlangen – nach dem Würfeln prüfen Sie ob das Zielfeld leer ist.
c) Beachten Sie nun auch Ketten von Leitern und Schlangen – bewegen Sie einen Spielstein nach dem Würfeln so lange, bis er weder auf einer Schlange noch einer Leiter landet.
d) Erweitern Sie ihr Spiel nun um den zweiten Spieler.

Erweitern Sie Ihr Spiel wie folgt:

- Bei einer 6 darf der Spieler noch einmal würfeln.
- Andere Spielfiguren werden geschlagen: Landet eine Spielfigur auf einem Feld, auf dem bereits eine gegnerische Spielfigur steht, muss diese 6 Felder zurück. (Dies kann auch durch das Benutzen einer Leiter oder einer Schlange passieren.)
- Das Ziel muss genau erreicht werden. Würde die Spielfigur auf einem höheren Feld (> 35) landen, bleibt sie auf dem aktuellen Spielfeld stehen.

Hinweis: Damit Sie bei der Programmierung besser den Überblick behalten, wo sich Ihre Spielsteine gerade befinden, wird mit diesem Blatt die Klasse `Spielfeld` verteilt, die folgende Methoden anbietet:

Methode	**Beschreibung**
`void paintField(int s1, int s2)`	Gibt das Spielfeld mit den Spielern auf den Positionen `s1` und `s2` aus; möchten Sie nur einen Spieler platzieren, dürfen Sie für den zweiten Spieler eine 0 übergeben.
`int dice()`	Würfelt eine Zahl zwischen 1 und 6
`void write(String output)`	Gibt eine Nachricht an den Benutzer aus

Rufen Sie die Methoden der Klasse `Spielfeld` auf, schreiben Sie also z.B. `Spielfeld.dice()`, um die Würfel zu werfen.

Aufgabe 5.3 [5 Punkte] (H) Jasmin II

Betrachten Sie die Klasse `Haus`. Geben Sie zu dieser den Callstack entsprechend der JASM.IN-Notation aus der Vorlesung an, wobei

a) der Nutzer als Hausbreite 19 angegeben hat und der oberste Aufruf auf dem Callstack die `main`-Methode in Zeile 58 ist.

b) der Nutzer als Hausbreite 8 und als Innenraumbreite 6 angegeben hat und

i) der oberste Aufruf auf dem Callstack die Methode `mauer()` in Zeile 37 ist, die for-Schleife in Zeile 33 zum letzten Mal durchaufen und die for-Schleife in Zeile 35 zum ersten Mal durchlaufen wird.

ii) der oberste Aufruf auf dem Callstack die Methode `mauer()` in Zeile 46 ist, die for-Schleife in Zeile 35 maximal oft durchlaufen wurde und die for-Schleife in Zeile 33 zum zweiten Mal durchlaufen wird.

Den Wert für die Variable `scanner` sollten Sie leer lassen.

```
1  import java.util.Scanner;
2
3  public class Haus {
4
5      //Breite des Hauses
6      static int h;
7      //Breite des Innenraums
8      static int i;
9
10   public static void dach () {
11
12     //Iteration ueber Zeilen, Dach ist halb so hoch wie Haus breit
13     for (int i = 0; i < h/2; i++) {
14         //Iteration ueber Spalten
15         for(int j = 0; j < h; j++){
16             if(j < (h/2)-i-1 || j > h-(h/2-i))
17                  //Ausgabe Leerzeichen
18                  System.out.print(' ');
19             else
20                 // Ausgabe Dach
21                 System.out.print('*');
22         }
23         System.out.println();
24     }
25   }
26
27   public static void mauer () {
28
29     //Balkenbreite
30     int m = (h-i)/2;
31
32     //Iteration ueber Zeilen, Dochhoehe ist konstant 6
33     for(int i = 0; i < 6; i++){
34         //Iteration ueber Spalten
35         for(int j = 0; j < h; j++){
36             //Ausgabe der Boden- und Deckenzeile
37             if(i == 0 || i ==  5)
38                 System.out.print('+');
39             else{
40                 if(j < m || j > h-m-1)
41                     System.out.print('+');
42                 else
43                     System.out.print(' ');
44             }
45         }
46         System.out.println();
47     }
48
49   }
50
51   public static void main (String[] args) {
52
53       Scanner scanner = new Scanner(System.in);
54
```

```
55        boolean repeat = true;
56
57        while(repeat){
58            System.out.println("Wieviele Zeichen soll das Haus breit sein? (
                  Die Anzahl der Zeichen muss gerade sein.)");
59            h = scanner.nextInt();
60            if(h > 1 && h % 2 == 0)
61                repeat = false;
62        }
63        repeat = true;
64        while(repeat){
65            System.out.println("Wieviele Zeichen soll der Innenraum breit
                  sein? (Die Anzahl der Zeichen muss gerade sein und "
66                      + "kleiner als " + h + ")");
67            i = scanner.nextInt();
68            if(i >= 0 && i < h && i % 2 == 0)
69                repeat = false;
70        }
71
72      dach();
73      mauer();
74    }
75  }
```

Hinweis: Wenn der oberste Aufruf auf dem Callstack die Methode in Zeile X ist, bedeutet das, dass die Zeile X noch nicht ausgeführt wurde.

Aufgabe 5.4 [11 + 2 Punkte] (H) Uno

Hinweis: In dieser Aufgabe gibt es zwei Bonuspunkte. Die entsprechenden Teilaufgaben müssen Sie also nicht bearbeiten, um die maximal mögliche Anzahl an regulären Punkten für die Hausaufgaben zu erreichen.

In dieser Aufgabe sollen Sie das Kartenspiel *Uno*[1] für zwei Spielerinnen implementieren. Bei Uno geht es darum, als erster alle Spielkarten abzulegen. Pro Runde darf eine Spielerin eine Karte ablegen, sofern diese nach Farbe oder Zahl zur zuletzt abgelegten Karte passt (zu Beginn des Spiels wird eine Karte vom Stapel gezogen und diese als zuletzt abgelegte Karte verwendet). Kann die Spielerin keine Karte ablegen, so muss sie eine weitere Karte vom Stapel ziehen. Das Spiel endet, wenn die erste Spielerin alle Karte ablegen konnte. Es gelten folgende weitere Regeln:

- Hat die Spielerin nach dem Ablegen lediglich noch eine Karte in der Hand, so muss sie durch Ausruf von *uno!* die übrigen Spielerinnen davor warnen, dass sie fast gewonnen hat. Vergisst sie diese Warnung oder gibt sie fälschlich ab, so erhält sie eine Strafkarte. In der Hausaufgabe soll dies implementiert werden, indem man der Benutzerin ermöglicht, bei der Wahl der nächsten abzulegenden Karte zusätzlich *uno!* zu rufen. Möchte die Benutzerin z.B. die dritte Karte im eigenen Blatt ablegen und dabei *uno!* rufen, so gibt sie `u3` statt lediglich `3` ein.
- Wer eine falsche Karte zu legen versucht, der erhält eine Strafkarte.
- Es gibt eine Reihe von Sonderkarten:
  - *Plus zwei* (farbig): Legt eine Spielerin diese Karte, so muss die nächste Spielerin zwei weitere Karten nehmen und wird übersprungen, außer diese legt selbst eine

[1] siehe `https://de.wikipedia.org/wiki/Uno_%28Kartenspiel%29`; gilt als Referenz, für die Implementierung der Hausaufgabe gilt nur die Aufgabenstellung

Plus zwei oder *Plus vier*; in diesem Fall wird die Anzahl der zu ziehenden Karten aufaddiert und die ursprüngliche Spielerin ist wieder an der Reihe. Der Vorgang kann so lange fortgesetzt werden, bis eine Spielerin keine *Plus zwei* oder *Plus vier* Karte mehr auslegen kann oder möchte; diese Spielerin muss die Anzahl der zu ziehenden Karten aufnehmen und darf keine weitere Karte spielen. Auf eine *Plus zwei* Karte kann jede andere Karte der passenden Farbe gelegt werden.

- *Plus vier* (schwarz, inkl. Farbwahl): Legt eine Spielerin A diese Karte, so muss die andere Spielerin B vier weitere Karten ziehen, außer sie verfährt wie bereits oben für die *Plus zwei* Karte beschrieben. Spielerin B wird nicht übersprungen, die Farbe wird allerdings von Spielerin A bestimmt. Die *Plus vier* Karte darf nur genau dann gelegt werden, wenn keine andere Karte passt.
- *Aussetzen* (farbig): Legt eine Spielerin diese Karte, so ist sie ein weiteres Mal dran. Es muss auf die Farbe der Karte geachtet werden.
- *Farbwahl* (schwarz): Legt eine Spielerin diese Karte, darf sie die Farbe wählen, mit der das Spiel fortfährt. Die nächste Spielerin darf nun lediglich eine Karte dieser Farbe oder eine schwarze Sonderkarte legen. Die *Farbwahl* Karte darf immer gelegt werden.

Für das Erreichen des *Bonus* soll das erstellte Programm nicht nur ein einziges Spiel, sondern eine Reihe von Spielrunden implementieren. Pro Runde erhält die Gewinnerin Punkte; die Punkte berechnen sich aus den übrigen Karten der Gegenspielerin:

- Eine Zahlenkarte zählt mit ihrem Wert.
- Eine farbige Sonderkarte zählt mit 10 Punkten.
- Eine schwarze Sonderkarte zählt mit 20 Punkten.

Die Benutzerin soll für den Bonus außerdem wählen dürfen, ob sie gegen den Computer oder gegen einen anderen Menschen spielen möchte. Der Modus für die zweite Spielerin soll einmal zu Beginn des Programmes erfragt werden und kann nicht mehr geändert werden. Sie dürfen sich selbst eine Strategie für den Computer überlegen; dieser muss aber mindestens so intelligent sein, dass ein Spiel gegen ihn prinzipiell sinnvoll möglich ist.

Gehen Sie bei der Implementierung dieser Aufgabe wie folgt vor:

a) Implementieren Sie zunächst die `zieheKarte()`-Methode.

b) Implementieren Sie nun ein einzige Spielrunde. Diese Aufgabe lässt sich wiederum in Teilschritten erledigen:

   i) Implementieren Sie zunächst den Spielablauf mit zwei menschlichen Spielern, wobei sie alle Sonderkarten ignorieren.

   ii) Implementieren Sie nun die Behandlung der farbigen Sonderkarten, also die der *Plus zwei* und *Aussetzen* Karte.

   iii) Implementieren Sie schließlich die Behandlung der übrigen, schwarzen Sonderkarten.

c) **Bonus**: Implementieren Sie das Management der Runden inklusive dem Zählen der Punkte entsprechend der übrig gebliebenen Karten des Verlierers nach jeder Runde.

d) **Bonus**: Implementieren Sie die Logik für den Computer-Spieler sowie die Möglichkeit, diesen statt Spieler zwei spielen zu lassen.

Verwenden Sie die Datei `src/uno-angabe/Uno.java` als Grundlage für Ihre Implementierung. In dieser Datei ist markiert, an welchen Stellen Sie Ihren Code einfügen sollen. Denken Sie auch daran, eigene Methoden zu erstellen, wo dies sinnvoll erscheint. Ihr Programm sollte desweiteren eine nachvollziehbare Struktur aufweisen; versuchen Sie daher zunächst, die Aufgabe zu lösen und verbessern Sie durch *Aufräumarbeiten* und Kommentare Ihren Code anschließend.

Sie müssen für die Lösung dieser Aufgabe einen Kartenstapel unbekannter Größe verwalten. Eine einzelne Karte wird durch zwei Buchstaben entsprechend der Beschreibung in `Uno.java` repräsentiert. Für einen Stapel von Karten verwenden Sie nun einen String, der mit steigender Kartenzahl wachsen darf. Um auf einzelne Karten innerhalb des Strings zugreifen zu können, beachten Sie, dass jeder String folgende Methoden bereitstellt:

Methode	**Beschreibung**
`String substring(int startIndex, int endIndex)`	Gibt einen Teilstring zwischen `startIndex` und `endIndex` zurück
`int charAt(int index)`	Gibt den Buchstaben am Index `index` im String zurück

Sie können eine der beschriebenen Methoden an einem String aufrufen, also z.B. `"hallo".substring(1, 2)`. Desweiteren steht ihnen die Methode `int random.nextInt(int max)` zur Verfügung, die Ihnen eine zufällig Zahl zwischen 0 und $max - 1$ liefert. Für eine genaue Beschreibung der Funktionsweise aller beschriebenen Methoden sollten Sie im Zweifel die Java-Dokumentation konsultieren.