Probeklausur: Einführung in die objektorientierte Programmierung mit Java 15WS

Alexander Bazo 26. Januar bis 2. Februar 2016

Allgemeine Hinweise

1. Die Bearbeitungszeit beträgt 60 Minuten. Sie könne 50 Punkte erreichen.
2. Verwenden Sie für das Lösen der Aufgaben die Programmiersprache Java. Alle gegebenen Code-Fragmente sind ebenfalls in Java verfasst. Halten Sie sich an die bekannten Regeln zur Codequalität.
3. Schreiben Sie Ihren Namen, Vornamen, Studiengang und Ihre Matrikelnummer leserlich unten auf jedes Angabenblatt bevor Sie mit der Bearbeitung beginnen! Blätter ohne diese Angaben werden nicht gewertet. Wenn Sie die Rückseite eines Blattes verwenden, notieren Sie dies bitte auf der Vorderseite. Kennzeichnen Sie eindeutig, zu welcher Aufgabe eine Lösung gehört. Verwenden Sie nur die ausgegebenen Angabenblätter für Ihre Lösung.
4. Benutzen Sie keine Bleistifte, keine rotschreibenden Stifte und kein TippEx (oder ähnliche Produkte).
5. Die Klausur ist als *Closed Book*-Klausur angelegt. Sie dürfen keine mitgebrachten Quellen oder Notizen zur Bearbeitung der Aufgaben verwenden. Technische Hilfsmittel sind ebenfalls nicht erlaubt.
6. Wenden Sie sich bei Unklarheiten in den Aufgabenstellungen immer an die Klausuraufsicht (Hand heben). Hinweise und Hilfestellungen werden dann, falls erforderlich, offiziell für den gesamten Hörsaal durchgegeben. Aussagen unter vier Augen sind ohne Gewähr.
7. Geben Sie keine mehrdeutigen (oder mehrere) Lösungen an. In solchen Fällen wird stets die Lösung mit der geringeren Punktzahl gewertet. Eine richtige und eine falsche Lösung zu einer Aufgabe ergeben also null Punkte.

Teil 1: Theorie

1) Variablen (2 Punkte)

Kreuzen Sie alle korrekten Aussagen an, die für Variablennamen in Java gelten.

- ☐ Variablennamen müssen mit einem Buchstaben oder Unterstrich beginnen
- ☐ Variablenname dürfen nur aus Buchstaben und Unterstrichen bestehen
- ☐ Variablennamen müssen mindestens aus drei Zeichen bestehen
- ☐ Variablennamen dürfen keine reservierten Wörter (z.b. public oder break) sein

2) Interfaces (2 Punkte)

Kreuzen Sie alle korrekten Aussagen an, die für Interfaces in Java gelten.

- ☐ Klassen können mehrere Interfaces implementieren
- ☐ Interfaces können Methodensignaturen und Konstanten enthalten
- ☐ Das Implementieren eines Interface wird durch das Schlüsselwort `extends` eingeleitet
- ☐ Als Programmierer können Sie selbst entscheiden, ob eine im Interface definierte Methode in einer Klasse, die dieses Interface implementiert, angelegt wird oder nicht

3) Vererbung (2 Punkte)

Welchen Zugriffsmodifikator darf eine Instanzvariable in einer Superklasse haben, damit mögliche Subklassen direkt auf diese geerbte Eigenschaft zugreifen könne?

- ☐ `public`
- ☐ `private`
- ☐ `protected`
- ☐ `hidden`

4) Codequalität (4 Punkte)

Nennen und Beschreiben Sie kurz jeweils ein Beispiel für eine gute und eine schlechte Benennung einer `boolean`-Variable, die angibt, ob sich eine Ellipse im sichtbaren Bereich der GraphicsApp-Anwendung befindet oder nicht.

Teil 2: Codeverständnis

Hinweise: Die Methode `System.out.println(String s)` gibt den String-Parameter `s` auf der Konsole aus. Ein Zeichen `(char)` an einer beliebigen Position einer String-Instanz kann über die Methode `charAt(int index)` der String-Klasse ausgelesen werden. Die Methode `indexOf(char c)` der String-Klasse gibt die (erste) Position des übergebenen Zeichens in der jeweiligen String-Instanz aus. Ist das Zeichen im String nicht vorhanden, wird `-1` zurückgegeben.

5) Codeanalyse (5 Punkte)

Geben ist die Methode `doStuff(String s1, String s2)`. Beschreiben Sie kurz den Aufbau der Methode und nennen Sie anschließend den wahrscheinlichen Einsatzzweck. Gehen Sie davon aus, dass der gegebene Code korrekt ist.

```
1  public String doStuff(String s1, String s2) {
2      String result = "";
3      for(int i = 0; i < s1.length(); i++) {
4          char c = s1.charAt(i);
5          if(s2.indexOf(c) != -1 && result.indexOf(c) == -1) {
6              result += c;
7          }
8      }
9      return result;
10 }
```


Name, Studiengang & Matrikelnummer:

6) Fehler finden (5 Punkte)

Finden Sie die Konventionsverstöße, Syntaxfehler und Bugs in der folgenden Methode. Markieren Sie die fehlerhaften Stellen im Code und beschreiben Sie den jeweiligen Fehler kurz unter Angabe der Zeilennummer. Schlecht gewählte Bezeichner für Methoden und Variablen sowie fehlerhafte Einrückungen zählen nicht als Fehler. Im Code sind fünf Fehler zu finden.

```
1  public class CharCounter {
2      private String inputString;
3
4      public CharCounter(String inputString) {
5          inputString = inputString;
6      }
7
8      public int countChar(char c) {
9          int NUMBER_OF_CHARS = 0;
10         for(int i = 0; i <= inputString.length(); i++) {
11             char currentChar = inputString.charAt(i);
12             if(currentChar = c) {
13                 NUMBER_OF_CHARS++;
14             }
15         }
16         return NUMBER_OF_CHARS
17     }
18
19 }
```


Name, Studiengang & Matrikelnummer:

Teil 3: Code implementieren

7) Pfade (10 Punkte)

Gegeben sind die Klassen `Point` und `BasicPath`. `Point` beschreibt einen einzelnen Punkt in einem zweidimensionalen Koordinatensystem, die Klasse `BasicPath` beschreibt einen Pfad aus mehreren Punkten (`Point`). Gehen Sie davon aus, dass dieser Code korrekt ist und beachten Sie die Hinweise am Ende der Aufgabenstellung (Seite 7).

```
1  public class Point {
2      private int x;
3      private int y;
4
5      public Point(int x, int y) {
6          this.x = x;
7          this.y = y;
8      }
9
10     public int getX() {
11         return x;
12     }
13
14     public int getY() {
15         return y;
16     }
17 }
```

```
1  public class BasicPath {
2      protected ArrayList<Point> points;
3
4      public BasicPath() {
5          points = new ArrayList<Point>();
6      }
7
8      public void addPoint(Point point) {
9          points.add(point);
10     }
11
12 }
```

7a) Implementieren Sie eine zusätzliche Klasse `ExtendedPath`. Diese erbt von `BasicPath` und verfügt über eine zusätzliche, öffentliche Methode `length`, die einen `double`-Wert zurückgibt. Dieser Wert beschreibt die Länge des Pfades, der sich aus den Teildistanzen zwischen der einzelnen Punkten in der `ArrayList` ergibt. Lagern Sie gegebenenfalls Teil der Lösung sinnvoll in zusätzliche Methoden aus.

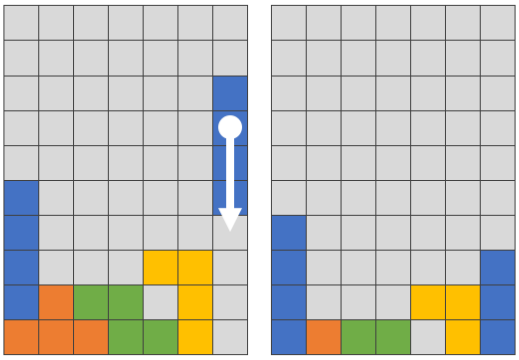
Für die Liste gilt: Die einzelnen Punkte sind in der korrekten Reihenfolge gespeichert. Eine Liste `[A,B,C]` beschreibt einen Pfad von `A` über `B` nach `C`. Die Länge des Pfads setzt sich aus den Teilstücken `A -> B` und `B -> C` zusammen. Ihre Lösung muss für Listen mit einer beliebigen Anzahl an Punkten funktionieren.

Für die Berechnung der Distanz `d` zwischen zwei Punkten gilt $d = \sqrt{dx^2 + dy^2}$. Dabei ist `dx` der Abstand (Differenz) der beiden x-Koordinaten und `dy` der Abstand der y-Koordinaten. In Java können Sie die Wurzel einer Zahl `a` mit der Methode `Math.sqrt(double a)` ziehen.

Name, Studiengang & Matrikelnummer:

8) Tetris (20 Punkte)

Gegeben ist die Klasse `Tetris`. Gehen Sie davon aus, dass dieser Code korrekt ist. Die Klasse stellt das Grundgerüst für das bekannte Spiel *Tetris* dar. Ziel dieses Spiel ist, fallende Spielsteine auf einem 2-dimensionalen Gitternetz so anzuordnen, das zusammengehörige, horizontale Linien entstehen. Diese Linien werden dann entfernt und alle weiteren Steine fallen entsprechen der Anzahl der entfernten Linien nach unten. Die nachstehende Grafik verdeutlicht dieses Prinzip.



Innerhalb des Spiels wird das Gitter durch ein mehrdimensionales `int`-Array abgebildet. Jede Zeile ist durch ein separates Array abgebildet, in dem jedes Element für eine einzelne Zelle steht. Durch die numerischen Werte `0` und `1` wird vermerkt, ob ein Feld frei (`0`) oder belegt (`1`) ist. Als *komplett* gilt eine Zeile dann, wenn alle ihre Elemente durch den numerischen Wert `1` repräsentiert werden, also belegt sind. Der erste Eintrag im **umschließenden** Array repräsentiert die oberste Zeile des Gitters, der letzte Eintrag die unterste Zeile.

```
Leeres Grid : {
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
}
```

```
Grid mit einigen Steinen : {
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0},
    {1,0,0,0,0,0,0,0,0,0},
    {1,0,0,0,0,0,0,0,0,0},
    {1,0,0,0,1,1,0,0,0,0},
    {1,1,1,1,0,1,0,0,0,0},
    {1,1,1,1,1,1,1,0,0,0},
}
```

Erweitern Sie den vorgegebenen Code, um die folgenden Teilaufgaben zu implementieren. Sie müssen nur die beschriebenen Probleme lösen. Sie können bei der Bearbeitung zusätzliche Methoden anlegen um Teilbereiche Ihrer Lösung auszulagern. Sie können sich in allen Teilaufgaben auf diese ausgelagerten Methoden beziehen. Gehen Sie bei der Lösung der Aufgabe davon aus, dass die Breite und Höhe des Gitters variabel sein kann.

```
1  public class Tetris {
2      private int[][] grid = {
3      {0,0,0,0,0,0,0,0,0,0},
4      {0,0,0,0,0,0,0,0,0,0},
5      {0,0,0,0,0,0,0,0,0,0},
6      {0,0,0,0,0,0,0,0,0,0},
7      {0,0,0,0,0,0,0,0,0,0},
8      {0,0,0,0,0,0,0,0,0,0},
9      {0,0,0,0,0,0,0,0,0,0},
10     {0,0,0,0,0,0,0,0,0,0},
11     {0,0,0,0,0,0,0,0,0,0},
12     {0,0,0,0,0,0,0,0,0,0},
13     {0,0,0,0,0,0,0,0,0,0},
14     {0,0,0,0,0,0,0,0,0,0}
15     };
16
17     private int removeCompleteLines() {
18         // Implementieren Sie hier Ihren Code
19     }
20
21     private void moveAllPieces(int lineOffset) {
22         // Implementieren Sie hier Ihren Code
23     }
24 }
```

9a) Implementieren Sie den Rumpf der vorgegeben Methode `removeCompleteLines`. Diese Methode entfernt alle kompletten Linien aus dem Gitter. Dabei werden, in kompletten Zeilen, alle Elemente von `1` auf `0` gesetzt. Die übrigen Zeilen des Gitters werden weder verändert noch verschoben. Die Methode gibt einen `int`-Wert zurück, der die Anzahl der entfernten Zeilen beinhaltet.

9b) Implementieren Sie den Rumpf der vorgegeben Methode `moveAllPieces`. Diese Methode bekommt die Position einer Gitter-Zeile als Parameter übergeben. Alle Zeilen **über** dieser Linie werden um eine Position nach unten verschoben.

Name, Studiengang & Matrikelnummer: