Wiederholungsklausur: Einführung in die objektorientierte Programmierung mit Java 15WS

Christian Wolff 23. September 2016

Allgemeine Hinweise

1. Die Bearbeitungszeit beträgt 60 Minuten. Sie können 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.
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. Das gilt auch für die *Multiple Choice*-Fragen. Kreuzen Sie bei einer Frage zwei richtige und zwei falschen Möglichkeiten an, ergibt das in der Summe null Punkte.

Teil 1: Theorie

1) Konstruktoren (2 Punkte)

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

- ☐ Konstruktoren können über das Schlüsselwort `new` aufgerufen werden.
- ☐ In einer Klasse können mehrere Konstruktoren definiert werden.
- ☐ In Konstruktoren einer Unterklasse kann der geerbte Konstruktor der Oberklasse immer nur als letzter Befehl aufgerufen werden.
- ☐ Beim Deklarieren eines Konstruktors kann ein beliebiger Name verwendet werden.

2) Funktionen und Methoden (2 Punkte)

Kreuzen Sie alle korrekten Aussagen an, die für Funktionen und Methoden in Java gelten.

- ☐ Bei jeder Deklaration einer Methode muss zwingend ein Zugriffsmodifikator angegeben werden.
- ☐ Im Rumpf einer Methode muss zwingend das Schlüsselwort `return` auftauchen.
- ☐ Nicht-statische Methoden werden immer im Kontext der Klassen-Instanz ausgeführt, zu der sie gehören.
- ☐ In einer Klasse können Methoden gleichen Namens durch unterschiedliche Parameter überladen werden.

3) Zugriffsmodifikatoren (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) Datenstrukturen (4 Punkte)

Sie schreiben ein Java-Programm, dass die Adressdaten mehrerer Personen speichert. Die Adressliste soll durchsuchbar sein. Zur Laufzeit werden neue Adressen bzw. Personen ergänzt. Jede Adresse besteht aus einem Namen, einer Straße, einer Hausnummer, einer Postleitzahl und einer Stadt. Beschreiben und begründen Sie kurz, wie Sie die einzelnen Adressen und wie Sie die gesamte Liste der Adressen sinnvoll in Ihrem Programm abbilden können. Sie müssen hier keinen eigenen Code schreiben.

Teil 2: Codeverständnis

Hinweise: Mit der Methode `indexOf(char c)` der `String`-Klasse lässt sich überprüfen, ob ein Zeichen `c` in einem String vorkommt oder nicht. Die Methoden wird auf der jeweiligen String-Instanz aufgerufen. Kommt das Zeichen im String vor, gibt die Methode die erste entsprechende Stelle im String als `int`-Wert zurück. Kommt das Zeichen nicht vor, wird `-1` zurückgegeben.

5) Codeanalyse (5 Punkte)

Gegeben 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 Klasse. 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 NumberFormatter {
2
3      public static final String defaultDelimiter = ",";
4      private String delimiter;
5
6      public NumberFormatter() {
7          this.delimiter = defaultDelimiter;
8      }
9
10     public NumberFormatter(String delimiter) {
11         delimiter = delimiter;
12     }
13
14     public int getFormattedNumber(int n) {
15         String tmp = String.valueOf(n);
16         String result = "";
17         for(int i = tmp.length(),j = 1; i >= 0; i--,j++) {
18             result = tmp.charAt(i) + result;
19             if(j%3 == 0 && i > 0) {
20                 result == this.delimiter + result;
21             }
22         }
23         return result;
24     }
25 }
```

Teil 3: Code implementieren

7) Suche im Bibliothekskatalog (15 Punkte)

Gegeben ist die Klassen `LibraryApp`, das Interface `Searchable` und das Enum `Type`. `LibraryApp` beschreibt eine rudimentäre Anwendung zur Verwaltung eines Bibliothekskatalogs. In der *ArrayList* werden die verschiedenen Einträge gespeichert. Die Methode `add` fügt einen neuen Eintrag zum Katalog hinzu. Die Methode `search` gibt eine `ArrayList` mit allen Einträgen zurück, die mit der übergebene Suchanfrage übereinstimmen. Gültige Einträge sind Instanzen aller Klassen, die das Interface `Searchable` implementieren. Dieses gibt zwei Methoden vor, um ein Objekt hinsichtlich einer Suchanfrage abzugleichen sowie um den Medientypen des gespeicherten Objekts abzufragen. Die Auswahl der möglichen Medientypen wird über das *Enum* `Type` vorgegeben.

```
1  public class LibraryApp {
2      private ArrayList<Searchable> libray;
3
4      public LibraryApp() {
5          libray = new ArrayList<Searchable>();
6      }
7
8      public void add(Searchable item) {
9          libray.add(item);
10     }
11
12     public ArrayList<Searchable> search(String query) {
13         ArrayList<Searchable> result = new ArrayList<Searchable
               >();
14         for(Searchable item: libray) {
15             if(item.match(query)) {
16                 result.add(item);
17             }
18         }
19         return result;
20     }
21 }
```

```
1  public interface Searchable {
2      public boolean match(String query);
3      public Type getType();
4  }
```

```
1  public enum Type {
2      BOOK
3  }
```

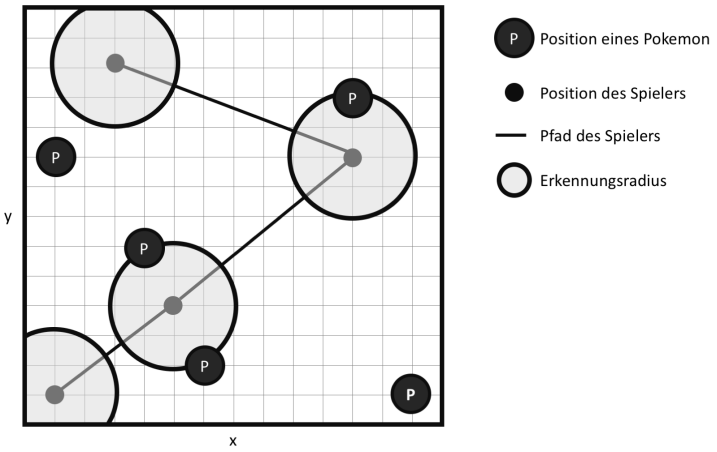
Aufgabe: Implementieren Sie eine zusätzliche Klasse `Book`. Diese implementiert das Interface `Searchable`. Die Klasse verfügt über zwei Instanzvariablen zum Speichern des Autoren und des Buchtitel. In einer weiteren Instanzvariable wird die Seitenzahl gespeichert. Alle drei Werte sollen im Konstruktor mit Hilfe von entsprechenden Parameter gesetzt werden. Verwenden Sie sinnvolle Bezeichner und Datentypen. Implementieren Sie anschließen die beiden Interface-Methoden. Die Methode `getType` soll den Wert `Book` aus dem Enum `Type` zurückgeben. Die Methode `match` gibt `true` zurück, wenn der als Parameter übergeben String `query` im Buchtitel und/oder im Autorennamen enthalten ist. Andernfalls wird `false` zurückgegeben.

Strings in Java: Mit Hilfe der Instanz-Methode `contains(CharSequence s)` der String-Klasse können Sie überprüfen, ob ein String die übergebene Zeichenkette enthält (Rückgabe: `true`) oder nicht (Rückgabe `false`). Als Parameter können Sie z.B. einen anderen String übergeben.

Name, Studiengang & Matrikelnummer:

8) Pokemon Go (15 Punkte)

In dieser Aufgabe implementieren Sie eine Komponente für das Spiel *Pokemon Go*. In diesem Spiel bewegt sich der Spieler über eine virtuelle Karte und kann Pokemon fangen, die sich in einem bestimmten Radius um ihn herum aufhalten. Für diese vereinfachte Variante des Spiels wird eine zwei-dimensionale Karten angenommen. Sowohl die Position des Spielers als auch die der Pokemon werden als Koordinatenpaare auf dieser Karte abgebildet. Befindet sich der Spieler etwa am unteren, linken Rand der Karte, ist seine Position `(x=0, y=0)`. Die nachstehende Grafik bildet die beschrieben Karte ab:



Für die Implementierung der Aufgabenstellung sind bereits einige Klassen vorgegeben. Betrachten Sie die dort gegebenen Funktionen als vollständig und funktionierend. Sie müssen keine Änderungen oder Ergänzungen an den vorgegeben Klassen durchführen. Für die genaue Verwendung der vorgegebene Methoden, können Sie sich auf die Kommentare im Code beziehen.

`PokemonHelper`: Diese Klasse dient Ihnen als Schnittstelle zu den Spieldaten. Sie können sich über die Methode `getCurrentPokemonOnMap` eine Liste aller Pokemon auf der Karte ausgeben lassen. Mit Hilfe der Methode `checkIfPokemonIsInCatchDistance` können Sie überprüfen, ob ein beliebiges Pokemon sich innerhalb einer bestimmten Distanz zum Spieler aufhält. Die Methode `hasNewPosition` gibt `true` zurück, wenn eine neue Spielerposition vorhanden ist. Ist dies der Fall, kann diese Position über die Methode `getNewPosition` ausgelesen werden. Ob anschließend eine weitere Position vorhanden ist, kann durch erneutes Abfragen (`hasNewPosition`) festgestellt werden.

`Pokemon`: Diese Klasse bildet ein Pokemon ab. Jedes Pokemon hat eine aktuelle Position und einen Namen. Beide Werte können per *Getter*-Methoden ausgelesen werden.

`Position`: Diese Klasse bildet eine Position auf der Karte ab. Jede Position besteht aus einem `x`- und einem `y`-Wert. Beide Werte lassen sich per *Getter*-Methoden ausle-

sen. Instanzen dieser Klasse werden zur Speicherung der Position von Spieler und Pokemon verwendet.
Erweitern Sie den vorgegebenen Code der Klasse `PokemonApp`, um die folgende Aufgabenstellung zu implementieren. Sie müssen nur die beschriebenen Probleme lösen. Sie können bei der Bearbeitung zusätzliche Methoden anlegen, um Teilbereiche Ihrer Lösung auszulagern.

Aufgabe: Erweitern Sie die Methode `walk` der Klasse `PokemonApp`. Verwende Sie die `PokemonHelper`-Instanz um solange neue Spieler-Positionen abzufragen, bis die entsprechende Methode `null` zurückliefert.
Für jede Spieler-Position soll Ihr Programm folgen Aufgaben erledigen.

- Prüfen Sie, ob sich Pokemon in der Nähe befinden, die gefangen werden können. Die maximale Distanz zum Fangen eines Pokemon wird in der Konstanten `MAX_CATCH_DISTANCE` angegeben.
- Befinde sich ein Pokemon in der Nähe, dass noch nicht gefangen wurde, soll es in die *ArrayList* `myPokemon` aufgenommen werden. Zusätzlich wird in diesem Fall der Text *Ein wildes POKEMON taucht auf. Du hast es gefangen.* ausgegeben. Dabei wird `POKEMON` mit dem Namen des jeweiligen Pokemon ersetzt.
- Befinde sich ein Pokemon in der Nähe, das bereits gefangen wurde, wird der Text *Ein wildes POKEMON taucht auf. Diese Art hast du schon gefangen.* ausgegeben. Dabei wird `POKEMON` mit dem Namen des jeweiligen Pokemon ersetzt.

```
1  public class PokemonApp {
2      private static final int MAX_CATCH_DISTANCE = 3;
3      private ArrayList<Pokemon> myPokemon;
4      private PokemonHelper helper;
5
6      public void walk() {
7          myPokemon = new ArrayList<Pokemon>();
8          helper = new PokemonHelper();
9
10         // Fuegen Sie hier Ihren eigenen Code ein. Sie koennen
               auch weitere Methode ergaenzen.
11
12     }
13
14 }
```

Textausgabe in Java: Mit der Methode `System.out.println(String s)` können Sie Text ausgeben.

```java
public class Pokemon {
    // ...
    public String getName() {
        return name;
    }
    public Position getCurrentPosition() {
        return currentPosition;
    }
}
```

```java
public class Position {
    // ...
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
}
```

```java
public class PokemonHelper {
    /* Gibt true zurueck, wenn eine neue Spielerposition
       verfuegbar ist.
     * Andernfalls wird false zurueckgegeben. */
    public boolean hasNewPosition() {/* ... */}

    /* Gibt die neue Spielerposition zurueck.
     * Wenn keine neue Position verfuegbar ist, wird null
       zurueckgegeben. */
    public Position getNewPosition() {/* ... */}

    /* Gibt eine ArrayList mit allen Pokemon zurueck, die
       aktuell auf der Karte zu finden sind. */
    public ArrayList<Pokemon> getCurrentPokemonOnMap() {/* ...
       */}

    /* Gibt true zurueck, wenn die Entfernung zwischen dem
       uebergebenen Pokemon pokemon und der uebergebenen
     * Position position nicht groesser als der Wert von
       maxCatchDistance ist. */
    public boolean checkIfPokemonIsInCatchDistance(Position
       position, Pokemon pokemon, int maxCatchDistance) {/* ...
       */}
}
```

Name, Studiengang & Matrikelnummer:

Name, Studiengang & Matrikelnummer: