Klausur zur Vorlesung: Einführung in die objektorientierte Programmierung mit Java im Wintersemester 2017/18

Alexander Bazo 21. Februar 2018

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) Listen (2 Punkte)

Nennen Sie einen Unterschied zwischen *Arrays* und *ArrayLists* in Java. Beschreiben Sie anhand dieses Unterschieds einen Anwendungsfall, in dem eine *ArrayList* einem *Array* vorzuziehen ist.

2) Fehler (4 Punkte)

Beschreiben und unterscheiden Sie die Begriffe Syntaxfehler, Bug und Exception.

3) Vererbung (2 Punkte)

Die Java-Klasse `A` verfügt über eine nicht-öffentliche (*private*) Variable *a*. Klasse `B` erbt von Klasse `A`. Kann die Eigenschaft *a* direkt in einer Methode von `B` ausgelesen oder mit einem Wert belegt werden? Begründen Sie Ihre Entscheidung.

4) Klassen (2 Punkte)

Welcher der folgenden Begriffe beschreibt keinen elementaren Teil von Klassen in Java?

- □ Methode
- □ Block
- □ Header
- □ Container

Teil 2: Codeverständnis

5) Codeanalyse (5 Punkte)

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

```
1 public static double doStuff(int x1, int y1, int x2, int y2) {
2     double dx = x2 - x1;
3     double dy = y2 - y1;
4     double result = Math.sqrt((dx * dx) + (dy * dy));
5     return result;
6 }
```

6) Fehler finden (5 Punkte)

Finden Sie die Konventionsverstöße, Syntaxfehler, Bugs und Logikfehler in dem folgenden Code-Abschnitt. 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 StringComparator {
2
3      private String TEXT;
4
5      public StringCompare(String text) {
6          this.TEXT = text;
7      }
8
9      public boolean hasEqualLengthAs(String input) {
10         if(TEXT.length() != input.length()) {
11             return true;
12         } else {
13             return false;
14         }
15     }
16
17     public String onlyUsesCharsFrom(String input) {
18         for(int i = 0; i < TEXT.length; i++) {
19             char c = TEXT.charAt(i);
20             if(input.indexOf(c) == -1) {
21                 return false;
22             }
23         }
24         return true;
25     }
26 }
```


Name, Studiengang & Matrikelnummer:

Teil 3: Code implementieren

7) Kursmanagment (15 Punkte)

In dieser Aufgabe implementieren Sie einige Klassen einer Software zur Verwaltung von Studierenden und deren Studienleistungen für den OOP-Kurs. Implementieren Sie - falls nicht anders angegeben - vollständige Klassen und achten Sie bei der Implementierung Ihres eigenen Codes auf sinnvolle Sichtbarkeiten, Datentypen und verständliche Bezeichner. Innerhalb der Anwendung wird eine einzelne Studienleistung durch Instanzen der Klasse `Assignment` (Siehe unten) abgebildet. Die Eigenschaft `grade` speichert die Note für diese Studienleistung und die Eigenschaft `weight` die Gewichtung, mit der diese in die Gesamtbewertung der Studienleistungen einfließt. Gehen Sie davon aus, dass der gegebene Code korrekt ist.

```
1  public class Assignment {
2
3      private double weight;
4      private double grade;
5
6      public Assignment(double weight, double grade) {
7          this.weight = weight;
8          this.grade = grade;
9      }
10
11     public double getGrade() {
12         return grade;
13     }
14
15     public double getWeight() {
16         return weight;
17     }
18
19 }
```

7a) Implementieren Sie eine Klasse `Student`. Diese repräsentiert einen einzelnen Studierenden und verfügt über private Eigenschaften zum Speichern des Namens und der Matrikelnummer. Alle Eigenschaften werden im Konstruktor gesetzt und können über öffentliche *Getter*-Methoden ausgelesen werden. Zusätzlich verfügt die Klasse über die Möglichkeit intern beliebig viele Studienleistungen (Instanzen der Klasse `Assignment`) in einer *ArrayList* zu speichern. Einzelne Studienleistungen können über eine öffentliche Methode zur Liste hinzugefügt werden; die gesamte Liste kann über eine weitere, öffentliche *Getter*-Methode ausgelesen werden.

7b) Implementieren Sie eine Klasse `RepeatingStudent`, die von `Student` erbt und einen Studierenden repräsentiert, der den Kurs wiederholt. Der Klasse werden im Konstruktor die Anzahl bereits erfolgter Kursbesuche sowie eine Liste der bereits erbrachten Studienleistungen übergeben und in passenden Instanzvariablen gespeichert. Die Anzahl aller Kursbesuche inklusive des aktuellen Besuchs soll über eine öffentliche Methode zugänglich gemacht werden.

7c) Ergänzen Sie in der Klasse `Student` eine öffentliche Methode `getGrade`, die die Gesamtnote der Studienleistungen unter Berücksichtigung der jeweiligen Gewichtungen und Einzelnoten als `double`-Wert zurückgibt. Die Gesamtnote ergibt sich aus der Summe der einzelnen Produkte von Gewichtung und Note der Einzelleistungen.

Name, Studiengang & Matrikelnummer:

7) Abseits (15 Punkte)

Vervollständigen Sie den gegebenen Code der Klasse `OffsetRule` um eine einfache Version der Abseitsregel beim Fussball zu implementieren. Ein Spieler wird dabei durch die Klasse `Player` abgebildet. Die Teamzugehörigkeit wird durch die Eigenschaft `team` der Klasse `Player` abgebildet. Diese kann nur den Wert `1` (*Team 1*) oder `2` (*Team 2*) annehmen. Eine List aller Spieler mit deren aktueller Position wird in der *ArrayList* `players` zur Verfügung gestellt. Das Spielfelld ist 800 Größeneinheiten breit. Die Torline von *Team 1* befindet sich immer auf der x-Position `0`, die des *Team 2* auf immer der x-Position `799`. Achten Sie bei der Implementierung Ihres eigenen Codes auf sinnvolle Sichtbarkeiten, Datentypen und verständliche Bezeichner. Halten Sie sich bei der Vervollständigung der Methoden an die vorgegebenen Signaturen. Gehen Sie davon aus, dass der gegebene Code korrekt ist.

> In diesem vereinfachen Beispiel ist die Abseitsregel wie folgt definiert: *Ein Spieler steht in einer Abseitsposition, wenn er sich in der gegnerischen Spielhälfte befindet und sich zwischen seiner Position und der gegnerischen Torlinie nicht mindestens zwei gegnerische Spieler befinden. Dabei gilt, das ein Spieler nicht im Abseits steht, wenn er sich auf der gleichen x-Position wie einer oder beide der relevanten gegnerische Spieler befindet.*

7a) Ergänzen Sie den Rumpf der Methode `getPlayerByNumber`, die den durch die Parameter `team` und `number` beschriebenen Spieler in der *ArrayList* `players` sucht und zurückgibt.

7b) Ergänzen Sie den Rumpf der Methode `getOpposingTeam`, die für den als Parameter übergebenen Spieler `player` eine Liste aller Spieler des gegnerischen Teams zurückgibt.

7c) Ergänzen Sie den Rumpf der Methode `isInOffsetPosition`, die `true` zurückgibt, wenn sich der als Parameter übergebene Spieler `player` gemäß der oben definierten Regel in einer Abseitsposition befindet. Die Methode gibt `false` zurück, wenn der Spieler sich nicht im Abseits befindet. Verwenden Sie die in Aufgabenteil *a* und *b* implementierten Methoden.

```java
public class Player {
    private int team;
    private int number;
    private int xPosition;

    public Player(int team, int number) {
        this.team = team;
        this.number = number;
        this.xPosition = 0;
    }
    public int getX() {
        return xPosition;
    }
    public int getTeam() {
        return team;
    }
    public int getNumber() {
        return number;
    }
}
```

```java
public class OffsetRule {
    // X-Position der Torlinien für Team 1 & Team 2
    private static final int TEAM_ONE_GOAL_LINE_X = 0;
    private static final int TEAM_TWO_GOAL_LINE_X = 799;
    // Liste mit Spielern und deren aktueller Position
    private ArrayList<Player> players;

    public OffsetRule(ArrayList<Player> players) {
        this.players = players;
    }

    private Player getPlayerByNumber(int team, int number) {
        // Ergänzen Sie hier Ihren Code
    }

    private ArrayList<Player> getOpposingTeam(Player player) {
        // Ergänzen Sie hier Ihren Code
    }

    public boolean isInOffsetPosition(int number, int team) {
        // Ergänzen Sie hier Ihren Code
    }
}
```


Name, Studiengang & Matrikelnummer: