Klausur zur Vorlesung: Einführung in die objektorientierte Programmierung mit Java im Wintersemester 2015/16

Christian Wolff 16. Februar 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) Arrays (2 Punkte)

Kreuzen Sie alle korrekten Aussagen an, die für Arrays in Java zutreffen.

- ☐ Arrays haben eine feste Länge, die nach Initialisierung nicht mehr verändert werden kann
- ☐ Arrays haben einen festen Datentyp, der für alle enthaltenen Elemente gilt
- ☐ Die Länge eines Arrays ist über die Methode `.size()` abrufbar
- ☐ Auf das erste Element eines Arrays wird mit dem Index `[1]` zugegriffen

2) Datentypen (2 Punkte)

Welche der folgenden Datentypen werden in Java nicht als primitive Datentypen abgebildet?

- ☐ `int`
- ☐ `String`
- ☐ `ArrayList`
- ☐ `boolean`

3) Debugging (4 Punkte)

Beschreiben Sie kurz den Unterschied zwischen einem Syntaxfehler und einem *Bug*.

4) Strings (2 Punkte)

Kreuzen Sie alle korrekten Aussagen an, die für Strings und Characters in Java zutreffen.

- ☐ Einzelne Zeichen können in Java über den primitiven Datentypen `char` abgebildet
- ☐ Die inhaltliche Übereinstimmung zweier Strings kann in Java eindeutig über den Operator == geprüft werden
- ☐ Ein String wird intern als Array aus `char`-Werten dargestellt
- ☐ In einer Variable vom Typen `char` können nur die Werte von `a` bis `z` und `0` bis `9` gespeichert werden

Teil 2: Codeverständnis

5) Codeanalyse (5 Punkte)

Geben ist die Methode `doStuff(String s1)`. 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 boolean doStuff(String s1) {
2     String[] words = {"time", "person", "year", "way", "day"};
3     for(int i = 0; i < words.length; i++) {
4         if(words[i].equals(s1)) {
5             return true;
6         }
7     }
8     return false;
9 }
```


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 CharReplacer {
2    private char TARGET_CHAR;
3    private char replacementChar;
4
5    public CharReplacer(char targetChar; char replacementChar) {
6      this.TARGET_CHAR = targetChar;
7      replacementChar = replacementChar;
8    }
9
10   public String replaceChar(String string) {
11     String result = "";
12     for(int i = 0; i <= string.length(); i++) {
13       char c = string.charAt(i);
14       if(c = TARGET_CHAR) {
15         result += replacementChar;
16       } else {
17         result += c;
18       }
19     }
20     return result;
21   }
22 }
```


Name, Studiengang & Matrikelnummer:

Teil 3: Code implementieren

7) Email (15 Punkte)

Gegeben ist die Klasse `Email` sowie das Interface `Attachment`. `Email` beschreibt eine Nachricht in einem *Mail-Client*. Das Interface `Attachment` wird von allen Klassen implementiert, die Dateien repräsentieren, die als Anhang einer Email beigefügt werden können.
Achten Sie beim Erstellen von Subklassen auf Sichtbarkeitsbereiche und Eigenschaften der Superklasse. Fügen Sie alle nötigen Komponenten hinzu und implementieren Sie, falls nicht explizit anders angegeben, komplette Klassen. Gehen Sie davon aus, dass der gegebene Code korrekt ist.

```java
public class Email {
    private String sender;
    private String recipient;
    private String subject;
    private String body;

    public Email(String sender, String recipient, String
       subject, String body) {
        this.sender = sender;
        this.recipient = recipient;
        this.subject = subject;
        this.body = body;
    }

    public void send() {
        if(!isEmpty()) {
            // Hier wird die Mail versendet
        }
    }
    // ...
}
```

```java
public interface Attachment {

    // Maximale Dateigröße des Anhangs in Kilobytes
    public static final long MAX_FILE_SIZE = 1024;

    // Gibt die Dateigröße des Anhangs in Kilobytes zurück
    public long getFileSize();

}
```

7a) Fügen Sie eine zusätzliche, öffentliche Methode `isEmpty` zur Klasse `Email` hinzu. Diese gibt den `boolean`-Wert `true` zurück, wenn **beide** String-Variablen `subject` und `body` einen leeren String enthalten. Sie müssen nicht die komplette Klasse abschreiben, es reicht, wenn Sie nur die neue Methode notieren.

7b) Entwerfen Sie eine neue Klasse `MailWithAttachment`, die von `Email` erbt und über eine zusätzliche, private Instanzvariable vom Typ `Attachment` verfügt. Ergänzen Sie eine öffentliche Methode ohne Rückgabewert, über die der Wert der Variable geändert werden kann. Die Methode bekommt den neuen Wert der Variable als Parameter übergeben. Wählen Sie einen passenden Namen für die neue Methode.

7c) Überschreiben Sie in `MailWithAttachment` die geerbte Methode `send`. Prüfen Sie in der Methode zuerst, ob der Anhang der Mail die erlaubte Dateigröße, die durch die Konstante im Interface `Attachment` angegeben wird, nicht überschreitet. Nur dann soll die geerbte Version der `send`-Methode aufgerufen werden. Übersteigt der Anhang das erlaubte Limit, wird die Methode direkt beendet.

Name, Studiengang & Matrikelnummer:

8) Fahrtenbuch (15 Punkte)

Gegeben sind die Klasse `LogbookEntry` und `Logbook`. Diese Klassen sind Teil eines elektronischen Fahrtenbuchs, mit dem Autofahrer berufliche und private Fahrten dokumentieren können. Ein Fahrtenbuch besteht dabei aus mehreren Einträgen, die die Distanz (Kilometer) und die Benzinkosten (Euro) der jeweiligen Fahrt abbilden. Zusätzlich wird für jeden Eintrag festgehalten, ob es sich um eine private oder berufliche Fahrt handelt. In dieser Aufgabe müsse Sie drei Methoden der `Logbook`-Klasse implementieren, deren Signaturen bereits vorgegeben sind. Achten Sie auf die erklärenden Kommentare im vorgegebenen Code. Gehen Sie davon aus, dass der gegebene Code korrekt ist.

Hinweis: Sie müssen nicht die komplette Klasse abschreiben, es reicht, wenn Sie nur die zu vervollständigenden Methoden notieren.

```
1  public class LogbookEntry {
2      // Gefahrene Distanz in Kilometern
3      private double distance;
4      // Benzinkosten für diese Strecke
5      private double costs;
6      // Gibt an, ob es sich um eine private Fahrt handelt
7      private boolean isPrivate;
8
9      public LogbookEntry(double distance, double costs, boolean
           isPrivate) {
10         this.distance = distance;
11         this.costs = costs;
12         this.isPrivate = isPrivate;
13     }
14
15     public double getDistance() {
16         return distance;
17     }
18
19     public double getCosts() {
20         return costs;
21     }
22
23     public boolean isPrivate() {
24         return isPrivate;
25     }
26 }
```

```java
import java.util.ArrayList;

public class Logbook {
    // Liste aller Fahrten
    private ArrayList<LogbookEntry> logbook;
    // Aktueller Tachostand in Kilometern
    private double mileage;
    // Durchschnittlicher Benzinverbrauch (Liter pro 100km)
    private double averageConsume;

    public Logbook(double mileage, double averageConsume) {
        this.mileage = mileage;
        this.averageConsume = averageConsume;
        logbook = new ArrayList<LogbookEntry>();
    }

    /**
     * Fügt einen neuen Eintrag zum Logbuch hinzu
     * @param distance Strecke in Kilometern
     * @param currentFulePrice Benzinpreis (Euro/Liter)
     * @param wasPrivateTrip Wahr, wenn Fahrt private war
     */
    public void addLog(double distance, double currentFulePrice
        , boolean wasPrivateTrip) {
        // ...
    }

    /**
     * Gibt die Gesamtdistanz aller beruflichen Fahren zurück
     * @return Berechnete Distanz in Kilometern
     */
    public double getTravelledDistanceForNonPrivateTrips() {
        // ...
    }

    /**
     * Gibt die Gesamtdistanz aller privater Fahren zurück
     * @return Berechnete Distanz in Kilometern
     */
    public double getTravelledDistanceForPrivateTrips() {
        // ...
    }

}
```


Name, Studiengang & Matrikelnummer:

8a) Vervollständigen Sie die Method `addLog` in `Logbook`. In dieser Methode wird ein neuer Eintrag zum Logbuch hinzugefügt. Nutzen Sie dazu die als Parameter übergebenen Werte. Zur Berechnung der Benzinkosten wird auch der Wert benötigt, der in der Instanzvariable `averageConsume` gespeichert ist. Zusätzlich zum Erstellen des neuen Eintrags soll auch der Tachostand im Fahrtenbuch angepasst werden.

Hinweis: Um den Benzinverbrauch zu berechnen, multiplizieren Sie die gefahrene Strecke mit dem durchschnittlichen Verbrauch **pro einem Kilometer**. Die tatsächlichen Kosten ergeben sich dann aus dem Produkt von Benzinverbrauch und aktuellem Benzinpreis.

8b) Vervollständigen Sie die Methoden `getTravelledDistanceForNonPrivateTrips` und `getTravelledDistanceForPrivateTrips` in `Logbook`. Diese Methoden geben die Gesamtdistanz aller beruflicher bzw. privater Fahrten zurück. Wenden Sie in dieser Aufgabe das Prinzip der *Dekomposition* an und lagern Sie Teilaufgaben wiederverwendbar in zusätzliche Methoden aus.