Prüfung in Objektorienter Programmierung

Hochschule Ravensburg-Weingarten
Prof. Dr. M. Zeller

Datum, Zeit	22. Juli, 2015, 10:30 - 12:00 Uhr (90 min)
Aufgabenblätter	11 Seiten (einschl. Deckblatt)
zugelassene Hilfsmittel	A (s. Prüfungsplan)

Studiengang	Raum
AI	C004, D002
EI	C004

Name: _____ Matrikelnr.: _____

Hinweise:

- Schreiben Sie bitte Name und Matrikelnummer auf jedes Aufgabenblatt.
- Schreiben Sie Ihre Lösung zu den Aufgaben auf den freien Platz, direkt anschließend an die Fragestellungen. Wenn Sie zusätzliche Blätter verwenden, so schreiben Sie bitte Name und Matrikelnummer auf jedes Blatt.
- Schreiben Sie lesbar!

Vom Prüfer auszufüllen:

Part	1	2	3	4	Summe
max.	15	5	20	14	54
Points					

Teil 1

1.1 (11 Punkte)

Analysieren Sie das Programm in Abschnitt **Constructors** der Arbeitsblätter "Programs and JDK-Documentation". Bitte beachten Sie bei den Antworten, dass mehr Zeilen als benötigt vorgegeben sein können.

Welche Ausgabe hat das Programm erzeugt, wenn es Zeile 45 beendet hat?
`SailingBoat mayFlower = new SailingBoat();`

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Welche zusätzliche Ausgabe hat das Programm erzeugt, wenn es Zeile 46 beendet hat?
`mayFlower.sail()`

. . . . . .
. . . . . .
. . . . . .

Welche zusätzliche Ausgabe hat das Programm erzeugt, wenn es Zeile 47 beendet hat?
`Watercraft bounty = new SailingBoat();`

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Welche zusätzliche Ausgabe hat das Programm erzeugt, wenn es Zeile 48 beendet hat?
`bounty.move()`

. . . . . .
. . . . . .
. . . . . .

1.2 (3 Punkte)

Die Definition der Klasse `Sails` wird wie folgt geändert:

```
class Sails {
    private float area = 30;
    Sails() {
        System.out.println("Creating Sails: ");
    }
    public float getArea(){
        return area;
    }
}
```

Ohne weitere Änderungen meldet der Compiler nun einen Fehler in der Klasse `SailingBoat`, Methode `sail()`. Welche Zeile(n) verursacht/verursachen den Fehler?

. . . . . .

Ändern Sie die Methode `sail()` so, dass sie sich übersetzen lässt und die gleiche Wirkung besitzt, wie die ursprüngliche Version. Verändern Sie die Klasse `Sails` nicht.

```
    void sail(){




    }
```

1.3 (1 Point)

Die Methode `main()` erhält einen zusätzlichen Methoden-Aufruf: `bounty.sail()`.

```
public static void main(String[] args) {
        SailingBoat mayFlower = new SailingBoat();
        mayFlower.sail();
        Watercraft bounty     = new SailingBoat();
        bounty.move();
        bounty.sail();    // <== new method call
    }
```

Lässt sich dieser Aufruf übersetzen und wenn ja, welche Ausgabe erzeugt er?

. . . . . .

. . . . . .

Teil 2

2.1 (5 Punkte) Exception Handling

Analysieren Sie das Programm in Abschnitt **Exception** der Arbeitsblätter "Programs and JDK-Documentation".

Das Programm definiert einige Unterklassen von Exception und eine Klasse `ExceptionTest`. Die Methode `foo()` kann jede der hier definierten Exceptions werfen, sowie eine Run-Time-Exception.

In welcher Reihenfolge müssen die Catch-clauses stehen, so dass jede oben erwähnte Art von Exception separat gefangen wird?

```
catch ( . . . . . ) {
      System.out.println(" --> cought xxxException: ");
```

Dabei steht xxx für "Runtime", "South", "NorthWest" und "North".

Tragen Sie hier die nötigen Catch-clauses ein:

```
try{
      int result = foo(1);
   }
```

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Teil 3

Analysieren Sie das Programm in Abschnitt **Playlist** der Arbeitsblätter "Programs and JDK-Documentation".

3.1 (4 Punkte)

Die Methode `void addSong(Song aSong)` der Klasse `PlayList` weist dem member `songList` einen neuen Wert zu (eine neue Referenz). Die Methode hängt ein Objekt des Typs `Song` an die PlayList an. Die bisher dort gespeicherte Referenz soll in dem neu angehängten Objekt gespeichert werden, so dass eine verkettete Liste ensteht. Die Datenstruktur, die das Programm erzeugt, ist in der Skizze neben dem Programmcode abgebildet. Die Objekte des Typs `Song` wurden dabei in folgender Reihenfolge angehängt: `Hip_0`, `Hop_1`, `Rap_2`, `Zap_3` und zuletzt `House_4`.

Vervollständigen Sie die Methode `void addSong(Song aSong)`:

```
public void addSong(Song aSong) {




}
```

3.2 (4 Punkte)

Die Methode `public Song removeSong()` entfernt das ersten Element der Liste `songList` und gibt das entfernte Element zurück. Ist die Liste leer, gibt die Methode `null` zurück.

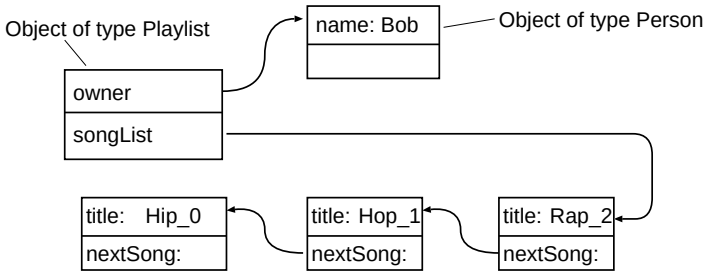
Vervollständigen Sie die Methode `public Song removeSong()`:

```
public Song removeSong() {




}
```

Die Datenstruktur nachdem zwei Elemente entfernt wurden:



3.3 (6 Punkte)

Die Methode `public void play()` der Klasse `PlayList` gibt den Namen des Besitzers der PlayList (member `owner`) aus. Außerdem gibt die Methode alle Titel der Stücke, die über den Member `songList` erreichbar sind, aus. Eine Ausgabe eines Programmlauf ist am Schluss des Abschnitts **Playlist** abgebildet.

Die Methode `public void play()` kann (soll) andere Methoden des Programms verwenden, sie darf aber die Datenstruktur nicht verändern.

Die Methode soll keine Exception werfen, wenn Sie auf eine null-Referenz stößt. Für eine null-Referenz erzeugt sie keine Ausgabe.

Vervollständigen Sie die Methode `public void play()`:

```
public void play() {
    System.out.println("\n ———— playing playlist ————" );
```

}

3.4 (6 Punkte)

Die Methode `public void play(int index)` der Klasse `PlayList` gibt den Titel eines Stückes der Liste `songList` aus. Für den Index-Wert n gibt die Methode den $n + 1$-ten Titel aus. In der Beispielausgabe in Abschnitt **Playlist** hat `index` den Wert drei. Entsprechend wird der Titel des vierten Stückes ausgegeben.

Die Methode darf die Datenstruktur des Programms nicht verändern. Die Methode soll keine Exception werfen, wenn der Wert von `index` kein gültiger Index ist (d. h. wenn er kleiner 0 oder zu groß ist). In diesem Fall erzeugt sie keine Ausgabe.

Vervollständigen Sie die Methode `public void play(int index)`:

```
public void play(int index) {
        System.out.println("\nplaying song " + index + " ————————");









}
```

Teil 4

Dieser Teil der Prüfung bezieht sich auf Abschnitt **Collection and IO** der Arbeitsblätter "Programs and JDK-Documentation".

Ein Programm schreibt Objekte von Typ `Customer` in eine Datei, liest Daten von dieser Datei und speichert Objekte in einer HashMap.

4.1 Ausgabe in eine Datei

4.1.1 (2 Punkte)

Die Methode `getDataOutputStream()` erzeugt einen gepufferten Ausgabe-Stream, um Daten in einer Datei zu speichern (Typ `DataOutputStream`). Das Programm beendet sich, wenn es den Stream nicht erzeugen kann.

Vervollständigen Sie die Methode `getDataOutputStream()`:

```
DataOutputStream getDataOutputStream(String fileName) {
    DataOutputStream dos = null;
    try {
        FileOutputStream fos = . . . . . .
        BufferedOutputStream bos = . . . . . .
        dos = . . . . . .
    } catch (IOException ioex) {
        ioex.printStackTrace();
    }
    if ( . . . . . ) {
        System.out.println("could not get OutputStream: " + fileName);
        System.exit(-1);
    }
    return dos;
}
```

4.1.2 (2 Punkte)

Die Methode `saveCustomer()` schreibt Daten eines Objekts vom Typ `Customer` auf einen `DataOutputStream`.

Vervollständigen Sie die Methode `saveCustomer()`:

```
void saveCustomer(Customer aCustomer, DataOutputStream dos) throws IOException {
    dos . . . . . .
    dos . . . . . .
    dos . . . . . .
}
```

4.2 Eingabe von einer Datei

4.2.1 (2 Punkte)

Die Methode `getDataInputStream()` erzeugt einen gepufferten Eingabe-Stream (Typ `DataInputStream`) um Daten aus einer Datei zu lesen. Das Programm beendet sich, wenn es den Stream nicht erzeugen kann. Vervollständigen Sie die Methode `getDataInputStream()`:

```
DataInputStream getDataInputStream(String fileName) {
    DataInputStream dis = null;
    try {
        FileInputStream fis = . . . . . .
        BufferedInputStream bis = . . . . . .
        dis = . . . . . .
    } catch (IOException ioex) {
        ioex.printStackTrace();
    }
    if ( . . . . . ) {
        System.out.println("could not get InputStream: " + fileName);
        System.exit(-1);
    }
    return dis;
}
```

4.2.2 (2 Punkte)

Die Methode `readCustomer()` liest Daten eines Objekts der Klasse `Customer` von einem `DataInputStream`. Nachdem sie die Daten eingelesen hat, erzeugt Sie ein Objekt der Klasse `Customer`. Das neue Objekt wird mit den eingelesenen Daten initialisiert. Wenn die Methode nicht alle Daten einlesen kann, gibt sie `null` zurück. Vervollständigen Sie die Methode `readCustomer()`:

```
Customer readCustomer(DataInputStream dis) {
    Customer aCustomer = null;
    try {
        int theId         = . . . . . .
        String theName    = . . . . . .
        float theVolume   = . . . . . .
        aCustomer         = . . . . . .
        System.out.println("id: " + theId);
    } catch (EOFException ex) {
        System.out.println("done reading file: ");
    }
    catch (IOException ioex) {
        ioex.printStackTrace();
    }
    return aCustomer;
}
```

4.3 Daten in einer HashMap speichern

4.3.1 (2 Punkte)

Die Klasse `PersonDB` definiert einen privaten Member `customerMap`. Dieser Member kann eine referenz auf eine HashMap aufnehmen, die Objekte vom Typ `Customer` speichert. Die HashMap verwendet als Schlüssel (`key`) Integer-Werte. Fügen Sie die Typ-Definition für die Variable `customerMap` ein und vervollständigen Sie den Aufruf zur Erzeugung der HashMap. Die Anfangskapazität der HashMap soll 100 betragen.

```
    . . . . . customerMap;  // provide proper data type
      :
      :
   void testCustomerIO() {
        custIO.writeCustomerFile(5, fileName);
        customerMap = new . . . . . // create HashMap
```

4.3.2 (2 Punkte)

Die Methode `readCustomerFile()` erhält einen Dateinamen und eine HashMap als Parameter. Sie öffnet einen Stream, liet Objekte des Typs `Customer` von diesem Stream und speichert diese Objekte in der HashMap. Der Wert des Memebers `customerId` dient jeweils als Schlüssel (key) für die HashMap.

Vervollständiegen Sie die Methode `readCustomerFile()`:

```
void readCustomerFile(String fileName, . . . . . customerMap) {
    try (DataInputStream dis = getDataInputStream(fileName)) {
        Customer aCustomer;
        while ((aCustomer = readCustomer(dis)) != null) {

            customerMap. . . . . ;  // store aCustomer
        }                           // in customerMap
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return;
}
```

4.3.3 (2 Punkte)

Die Methode `testCustomerIO()` speichert einige Objekte in einer HashMap, greift dann wieder auf diese Objekte zu und gibt sie auf dem Bildschirm aus. Wenn das Programm die Zeile 124 ereicht hat, sind in der HashMap Objekte mit folgenden Schlüsseln eingetragen: 70529, 48372 und 35830.

```
118     void testCustomerIO() {
119             :
120             :
121         custIO.readCustomerFile(fileName, customerMap);
122             :
123             :
124         Customer aCustomer = custIO.createRandomCustomer();
125         aCustomer.print();
126         customerMap.put(12345, aCustomer);
127         aCustomer = custIO.createRandomCustomer();
128         aCustomer.print();
129         customerMap.put(12345, aCustomer);
130         aCustomer = customerMap.get(12345);
131         aCustomer.print();
132         aCustomer = customerMap.get(1111);
133     }
```

Die Ausgabe des Programms beginnt wie folgt:

Ausgabe von Zeile 125:

```
ID: 61960, Name: Don, Revenue: 4774.1045
```

Ausgabe von Zeile 128:

```
ID: 86494, Name: Chris, Revenue: 4137.799
```

Welche Ausgabe erzeugt Zeile 131:

. . . . . .

Was passiert in Zeile 132: