TU Darmstadt — FB Informatik — Prof. J. Fürnkranz

Vordiplomsklausur - Lösungsvorschlag

Allgemeine Informatik II – SS 2005

15.09.2005 – 11:30 - 13:30 Uhr

Hinweise:

- Als Hilfsmittel ist nur ein schwarzer oder blauer Schreibstift erlaubt!
- Füllen Sie das Deckblatt vollständig aus!
- Schreiben Sie auf jedes Aufgabenblatt Ihren Namen und Matrikelnummer!
- Schreiben Sie ihre Lösung in die vorgesehenen Zwischenräume oder auf die Rückseite des jeweiligen Aufgabenblattes!

Nachname	
Vorname	
Matrikelnummer	
Fachbereich	
Studiengang	

Aufgabe	1	2	3	4	5	6	$\sum$
Punkte							
Maximum	8	15	8	21	30	18	100

1 IO und Exceptions (8 Pkte.)

Gegeben sei folgende Methode:

```
1  public String getTextLines(FileReader fReader) {
2     StringBuffer buffer = new StringBuffer();
3    String line;
4
5     BufferedReader bReader = new BufferedReader(fReader);
6
7     // ...
8
9     while ((line = bReader.readLine()) != null) {
10       buffer.append(line);
11    }
12
13    // ...
14
15    return buffer.toString();
16 }
```

Die Funktion `readLine()` in Zeile 9 kann eine Ausnahme vom Typ `IOException` auslösen.

1. **(5 Pkte.)** Ergänzen Sie an den markierten Stellen (`// ...`), so dass, falls eine `IOException` auftritt, die Meldung „Fehler beim Einlesen“ auf dem Bildschirm ausgegeben wird.

   Lösungsvorschlag:
   Zeile 9 - 11 in obigem Code müssen in einen try-catch-Block eingebettet werden.

   ```
   1  try {
   2
   3     while ((line = bReader.readLine()) != null) {
   4        buffer.append(line);
   5     }
   6
   7  } catch (IOException e) {
   8     System.out.println("Fehler beim Einlesen aufgetreten");
   9  }
   ```

2. **(3 Pkte.)** Welche Zeile muss wie geändert werden, wenn die Exception nicht gefangen sondern an den Aufrufer der Methode weitergeleitet werden soll? Schreiben Sie die geänderte Zeile auf.

   Lösungsvorschlag:
   Der Methodenkopf (Zeile 1) muss um `throws IOException` erweitert werden.

   ```
   1  public String getTextLines(FileReader fReader) throws
      IOException { ...
   ```

   Ein try-catch-Block ist dann nicht erforderlich.

2 Rekursion (15 Pkte.)

Gegeben sei folgende Klassendefinition:

```
1  public class Klaas {
2     public static int x (int n) {
3        if (n > 0)
4           return 2*x(n-1);
5        else
6           return 1;
7     }
8
9     public static void y (int m, int n) {
10       if (m > 0) {
11          System.out.print(x(n-m) + " ");
12          y(m-1,n);
13          System.out.print(" " + x(n-m));
14       }
15       else {
16          System.out.print(x(n));
17       }
18    }
19 }
```

1. **(3 Pkte.)** Geben Sie eine nicht-rekursive Formel an, die ausdrückt, was die Methode `x` für eine gegebene positive Zahl `n` berechnet.

   Lösungsvorschlag:
   Es wird 2^n berechnet.

2. **(4 Pkte.)** Was wird von diesem Programm am Bildschirm ausgegeben, wenn die Methode `y` mit den Werten $m = 4$ und $n = 4$ aufgerufen wird.

   Lösungsvorschlag:
   1 2 4 8 16 8 4 2 1

3. **(2 Pkte.)** Ergänzen Sie unten stehendes Programmstück um einen entsprechenden Aufruf der Methode `y` mit den Werten $m = 4$ und $n = 4$.

```
1  public class Klever {
2     public static void main (String[] args) {
3        // Aufruf von y folgt hier
4
5
6
7     }
8  }
```

   Lösungsvorschlag:
   Sowohl `Klaas.y(4,4);` als auch `Klaas k = new Klaas(); k.y(4,4);` oder ähnliches ist richtig.

4. **(3 Pkte.)** Würde das Programm `Klever` auch noch funktionieren, wenn man

- in der Methode `Klaas.x` `public` durch `private` ersetzt
- in der Methode `Klaas.y` `public` durch `private` ersetzt

Geben Sie eine kurze Begründung.

Lösungsvorschlag:
Bei `x` würde es funktionieren, da `x` nur von der eigenen Methode `y` aufgerufen wird, bei `y` nicht, da eine `private` Methode nicht ausserhalb der Klasse aufgerufen werden kann.

5. **(3 Pkte.)** Würde das Programm auch noch funktionieren, wenn man in Zeile 4 den Aufruf `x(n-1)` durch `this.x(n-1)` ersetzen würde? Begründung?

Lösungsvorschlag:
Nein, da `this` ein Objekt instantiiert, `x` eine Klassenmethode ist, und eine Klassenmethode nicht mit einem Objekt aufgerufen werden kann.

3 Multiple choice (8 Pkte.)

Kreuzen Sie an, ob folgende Aussagen wahr oder falsch sind.

Jede richtige Antwort zählt einen Punkt. Für jede falsche Antwort wird ein Punkt abgezogen, Sie können aber in Summe nicht weniger als 0 Punkte auf dieser Aufgabe erreichen. Fehlende Antworten werden nicht gewertet.

Wahr	**Falsch**	
☐	☒	Ein Interface kann Objekt-Methoden definieren und weitervererben.
☐	☒	Eine abstrakte Klasse darf Klassen-Methoden aber keine Objekt-Methoden definieren.
☐	☒	Eine Konstante wird mit dem Schlüsselwort `static` definiert.
☒	☐	Eine Konstante muß sofort initialisiert werden.
☒	☐	Das Zugriffsrecht `protected` vor einer Variable legt fest, dass diese Variable nur von Methoden der Klasse oder einer Unterklasse gelesen oder überschrieben werden darf.
☐	☒	Mit `private` markierte Variablen dürfen von außerhalb der Klasse definierten Methoden gelesen, aber nicht überschrieben werden.
☐	☒	Eine Methode, die eine Exception wirft, muß immer in einem `try-catch` Konstrukt aufgerufen werden.
☐	☒	Jedem `try`-Block muß *genau* ein `catch`-Block folgen.

4 Body-Mass-Index (21 Pkte.)

Ein Fitness-Studio möchte seinen Mitgliedern eine Ernährungsberatung anbieten. Dazu sollen gezielt alle Mitglieder angesprochen werden, die an Übergewicht leiden. Bewertet wird das Gewicht nach dem Body-Mass-Index (BMI). Ein Wert von 20-25 bedeutet Normalgewicht. Werte darüber bedeuten Übergewicht.

Formel: $BMI = \frac{Koerpergewicht}{(Koerpergroesse\ in\ m)^2}$

Gegeben Sei die folgende Klasse:

```
1 public class FitnessStudioMitglied {
2         String name;
3         int gewicht;
4         int groesse;
5         FitnessStudioMitglied next;
6 }
```

Sie erhalten eine korrekt aufgebaute Liste von `FitnessStudioMitglieder`-Objekten. Die Varíable `next` zeigt auf das nächste Objekt, am Ende der Liste hat sie den Wert `null`.

1. **(3 Pkte.)** Schreiben Sie eine `static` Methode, die ein `FitnessStudioMitglied`-Objekt übergeben bekommt und dessen BMI-Wert zurückgibt.

2. **(9 Pkte.)** Schreiben Sie eine `static` Methode, die ein `FitnessStudioMitglied`-Objekt übergeben bekommt und mit einem *rekursiven* Algorithmus alle Namen der Personen ausgibt, die an Übergewicht leiden – bewertet nach dem BMI. Sie dürfen bei dieser Lösung keine Schleifenkonstrukte verwenden. Verwenden Sie sinnvoll auch bereits definierte Methoden.

3. **(9 Pkte.)** Schreiben Sie eine `static` Methode, die ein `FitnessStudioMitglied`-Objekt übergeben bekommt und nach einem *iterativen* Verfahren alle Namen der Personen ausgibt, die an Übergewicht leiden. Sie dürfen bei dieser Lösung keine Methodenaufrufe verwenden. **Ausnahme:** Der Befehl zur Ausgabe auf dem Bildschirm und eine Funktion zur Berechnung des BMI.

Lösungsvorschlag:

```
1  public static int berechneBMI(FitnessStudioMitglied mitglied) {
2          return mitglied.gewicht / (mitglied.groesse * mitglied.
             groesse);
3  }
4
5  public static void printRekursiv(FitnessStudioMitglied mitglied) {
6          if (mitglied != null) {
7                  if (berechneBMI(mitglied) > 25) {
8                          System.out.println(mitglied.name);
9                  }
10                 printRekursiv(mitglied.next);
11         }
12 }
13
14 public static void printIterativ(FitnessStudioMitglied mitglied) {
15         FitnessStudioMitglied item = mitglied;
16         while (item != null) {
```

```
17                      if (berechneBMI(item) > 25) {
18                              System.out.println(item.name);
19                      }
20                      item = item.next;
21              }
22      }
```

5 Lotto-Simulator (30 Pkte.)

Programmieren Sie einen Lottozahlen-Simulator, der eine Ziehung m aus n durchführt (bei 6 aus 49 ist $m = 6$ und $n = 49$). m Zahlen werden im Bereich von 1 bis n per Zufall generiert und *sortiert* ausgegeben. Jede Zahl aus dem angegebenen Bereich kann nur einmal gezogen werden.

Gehen Sie bei der Implementierung schrittweise vor:

1. Definieren Sie eine Klasse `LottoSimulator`. Die gezogenen Zahlen sollen als eine interne Datenkomponente `ziehung` abgespeichert werden. Die Zahlen m und n sollen ebenfalls gespeichert werden. Auf alle Datenkomponenten dürfen nur Methoden der Klasse `LottoSimulator` zugreifen.

2. Schreiben Sie einen Konstruktur **public** LottoSimulator(**int** m, **int** n) für diese Klasse, der das Objekt mit den angegeben Dimensionen initialisiert.

3. Schreiben Sie eine Methode **public** Set generateLottozahlen(). Diese soll m Lotto-Zahlen zufällig erzeugen, intern in `ziehung` speichern, und als Ergebnis zurückliefern.

4. Schreiben Sie eine Methode **public int** checkTip(Set tip), die als Ergebnis zurückgibt, wie viele der als `tip` übergebenen Zahlen gezogen wurden.

Hinweise:
Im Package `java.util` existieren mehrere Klassen, die bei der Implementierung hilfreich sein könnten:

Random Die Methode **public int** nextInt (**int** n) der Klasse `Random` erzeugt eine zufällige Zahl im Bereich von 0 und $n - 1$. Die Klasse hat einen Default-Konstruktr `Random()`.

Beachten Sie, dass der Zufallszahlengenerator natürlich auch Zahlen erzeugt, die schon in der Menge vorhanden sind.

TreeSet implementiert das Interface `Set` in der Form einer nach natürlicher Ordnung (bei Zahlen aufsteigende Reihenfolge) sortierte Menge von Elementen. Die wichtigsten Methoden der Interfaces `Set` und `Iterator` sollten Ihnen bekannt sein.

Vergessen Sie nicht, die verwendeten Packages korrekt zu importieren!

Lösungsvorschlag:

```
1  import java.util.Iterator;
2  import java.util.Random;
3  import java.util.Set;
4  import java.util.TreeSet;
5
6  //oder import java.util.*;
7
8  public class LottoSimulator {
9
10   //Zufallszahlengenerator
11   private Random r;
12
13   private int m;
14   private int n;
15
```

```
16    private Set ziehung;
17
18    // Konstruktor
19    public LottoSimulator(int m, int n) {
20      this.m = m;
21      this.n = n;
22      r = new Random();
23      ziehung = new TreeSet();
24    }
25
26    public Set generateLottozahlen() {
27       //ziehung leeren/initialisieren
28       ziehung.clear()
29
30       //Lottozahlenmenge füllen bis sechs verschiedene Zahlen drin
           sind.
31       while (ziehung.size() < m) {
32         ziehung.add(new Integer(r.nextInt(n) + 1));
33       }
34
35       return ziehung;
36    }
37
38    public int checkTip(Set tip) {
39      int treffer = 0;
40
41      Iterator tipIter = tip.iterator();
42
43      while (tipIter.hasNext()) {
44        if (ziehung.contains(tipIter.next())) {
45          treffer++;
46        }
47      }
48
49      return treffer;
50    }
51
52  }
```

6 Abstrakte Klassen und Interfaces (18 Pkte.)

Ein grosser Werkzeughändler verkauft seine Produkte nicht nur, sondern manche Produkte kann man auch mieten. Jedem Produkt ist daher ein Preis zugeordnet, manchen Produkten auch ein Mietpreis. Die Art und Weise, wie der Preis und der Mietpreis berechnet werden, ist von Produkt zu Produkt verschieden.

1. **(3 Pkte.)** Definieren Sie eine abstrakte Klasse `Produkt`, in der Sie festlegen daß jedes `Produkt` eine Methode `getPreis` hat, die den Preis des Produktes zurückgibt, und eine Methode `getName`, die den Namen des Produktes liefert.

   Lösungsvorschlag:

```
1 public abstract class Produkt {
2    public abstract int getPreis();
3    public abstract String getName();
4 }
```

2. **(2 Pkte.)** Definieren Sie ein Interface `Mietbar`, das für die Objekte, die es implementieren, festlegt, daß eine Methode `int getMietPreis()` vorhanden sein muß.

   Lösungsvorschlag:

```
1 public interface Mietbar {
2    int getMietPreis();
3 }
```

3. **(5 Pkte.)** Definieren Sie eine Klasse `Gabelstapler` als Unterklasse von `Produkt` und Implementierung von `Mietbar`. Der Preis eines Gabelstaplers soll 10,000 Euro betragen, der Mietpreis 100 Euro.

   Lösungsvorschlag:

```
1 public class Gabelstapler extends Produkt implements Mietbar {
2    public String getName() {return "Gabelstapler";}
3    public int getPreis() {return 10,000;}
4    public int getMietPreis() {return 100;}
5 }
```

4. **(8 Pkte.)** Schreiben Sie eine Methode **public void** printMietbar(Produkt[] produkte), die aus einem Array von Produkten die Namen aller mietbaren Produkte auf den Bildschirm schreibt.

   Lösungsvorschlag:

```
1 public void printMietbar(Produkt[] produkte) {
2    for (int i = 0; i < produkte.length; i++) {
3       if (produkte[i] instanceof Mietbar) {
4          System.out.println(produkte[i].getName());
5       }
6    }
7 }
```