TU Darmstadt — FB Informatik — Prof. J. Fürnkranz

Semestralklausur - Lösungsvorschlag

Allgemeine Informatik II – SS 2005

20.07.2005 – 09:30 - 11:30 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	10	14	8	18	20	30	100

1 Collections (10 Pkte.)

In der Vorlesung haben Sie verschiedene Interfaces kennengelernt, die es erlauben, Sammlungen (Collections) von Objekten zu repräsentieren.

1. **(6 Pkte.)** Sie haben die Aufgabe, die Verwaltung eines on-line Geschäfts zu programmieren. Welches Interface `Set`, `List`, oder `Map` würden Sie zur Repräsentation der folgenden Daten verwenden. Begründen Sie jeweils Ihre Antworten!

   a) Alle möglichen Arten von Artikeln, die man im Geschäft kaufen kann.

   b) Alle Preise für alle Artikel.

   c) Einen Einkaufskorb, d.h. eine Sammlung aller Artikel, die bei einem Kaufvorgang gekauft werden (ein Artikel kann natürlich auch mehrmals gekauft werden).

   Sie können annehmen, dass entsprechende Datenstrukturen für einzelne Kunden, Artikel, etc. bereits vordefiniert sind, es geht nur um die Verwendung eines geeigneten Interfaces.

   Lösungsvorschlag:
   Folgende Interfaces bieten sich an:

   a) `Set` – Da es jede Artikelart nur einmal gibt und nicht mehrmals.

   b) `Map` – Zuordnung von Artikelart zu Preis.

   c) `List` – Da eine Artikelart ja mehrmals im Einkaufskorb vorhanden sein kann.

2. **(2 Pkte.)** Mit der Methode `keySet()` erhalten Sie für eine Klasse, die das Interface `Map` implementiert, ein `Set` aller Schlüssel zurück. Erklären Sie, warum hier als Rückgabe-Typ `Set` und nicht `List` gewählt wurde.

   Lösungsvorschlag:
   Der Rückgabe-Typ `Set` macht deshalb Sinn, da es jeden Schlüssel in der `Map` nur einmal gibt.

3. **(2 Pkte.)** Erklären Sie kurz die Idee und Funktionsweise einer Hash-Tabelle.

   Lösungsvorschlag:
   Eine Tabelle, die von dem Schlüssel(Key)-Objekt einen numerischen Hash-Wert berechnet, der die Position des zugeordneten Wertes beschreibt.

2 Klassen (14 Pkte.)

Gegeben sei folgende Klassendefinition:

```
1  class Rec {
2     void x (int n) {
3        System.out.print("0");
4        if (n > 0)
5           x(n-1);
6     }
7     static void z (int n) {
8        x(n);
9        System.out.println();
10    }
11    static void y (int n) {
12       z(n);
13       if (n > 0)
14          y(n-1);
15       z(n+1);
16    }
17 }
```

1. **(1 Pkte.)** y ist eine Klassen-Methode. Woran erkennt man das?

   Lösungsvorschlag:
   Eine Klassen-Methode erkennt man am Schlüsselwort **static**.

2. **(2 Pkte.)** Sie wollen in einem Programmstück einer Klasse im selben Package die Klassen-Methode y aufrufen, und dabei die Zahl 5 als Parameter übergeben. Schreiben Sie einen enstprechenden Aufruf ohne dabei ein Objekt anzulegen.

   Lösungsvorschlag:
   Aufruf: `Rec.y(5)`

3. **(2 Pkte.)** Wäre folgender Aufruf ebenfalls möglich?

   ```
   1 Rec r = new Rec();
   2 r.y(9);
   ```

   Begründen Sie!

   Lösungsvorschlag:
   Ja, da Klassenmethoden auch aus Objektmethoden aufgerufen werden können.

4. **(2 Pkte.)** Obige Klassen-Definition enthält einen Fehler, der bereits zur Compile-Zeit erkannt wird. Welchen? Wie können Sie den Fehler korrigieren?

   Lösungsvorschlag:
   Fehler in Zeile 8 - Aufruf einer nicht-static Methode.
   Zur Fehlerkorrektur ist ein static in der Methodendeklaration in Zeile 2 zu ergänzen.

5. **(7 Pkte.)** Schreiben Sie die Ausgabe auf, die nach Behebung des Fehlers produziert wird, wenn der Methode y der Wert 5 übergeben wird.

   Hinweis: *Sie können diese Frage auch beantworten, wenn Sie den Fehler im vorigen Punkt nicht gefunden haben.*

   Lösungsvorschlag:
   Folgendes Muster entsteht:

```
000000
00000
0000
00
0
00
000
0000
00000
000000
0000000
```

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**	
☒	☐	Eine abstrakte Klasse kann Objekt-Methoden definieren und weitervererben.
☐	☒	Eine abstrakte Klasse darf keine Konstanten definieren.
☐	☒	Ein Interface darf Klassen-Methoden aber keine Objekt-Methoden definieren.
☐	☒	Das Zugriffsrecht `protected` vor einer Methode legt fest, dass diese Methode nur von Methoden einer Unterklasse überschrieben werden darf.
☒	☐	Das Zugriffsrecht `private` vor einer Methode legt fest, dass nur Methoden derselben Klasse diese Methode verwenden dürfen.
☒	☐	Mehrere Exceptions verschiedenen Typs können mit einer einzigen `catch`-Klausel gefangen werden.
☐	☒	Exceptions vom Typ `RunTimeException` müssen gefangen werden.
☐	☒	Eine Variable, die als `static` deklariert wurde, darf in einer Unterklasse nicht verändert werden.

4 Rekursiv vs. iterativ (18 Pkte.)

Die Klasse `PersonItem` sei wie folgt definiert:

```
1 public class PersonItem {
2     String name;
3     int alter;
4     PersonItem next;
5 }
```

Gehen Sie davon aus, dass sie eine korrekt aufgebaute Liste von `PersonItem`-Objekten haben. Die Variable `next` zeigt also auf das nächste `PersonItem`-Objekt in der Liste. Am Ende der Liste hat die Variable `next` den Wert `null`.

Ihre Aufgabe ist es, eine Java-Methode zu schreiben, die alle Namen der Personen zeilenweise auf dem Bildschirm ausgibt, die älter sind als ein vorgegebener Wert.

1. **(9 Pkte.)** Schreiben Sie eine Methode **static void** printNames_recursive(PersonItem start, **int** age), die rekursiv alle Namen der Personen ausgibt, die älter sind, als der Wert des Parameters age. Sie dürfen bei dieser Lösung keine Schleifenkonstruktionen verwenden.
2. **(9 Pkte.)** Schreiben Sie eine Methode **static void** printNames_iterativ(PersonItem start, **int** age), die iterativ alle Namen der Personen ausgibt, die älter sind, als der Wert des Parameters age. Sie dürfen bei dieser Lösung keine Methodenaufrufe verwenden. **Ausnahme:** Der Befehl zur Ausgabe auf dem Bildschirm.

Tip: Sie dürfen annehmen, dass das an ihre Methoden übergebene `PersonItem`-Objekt nicht `null` sein wird.

Lösungsvorschlag:

```
1  static void printNames_recursive(PersonItem start, int age) {
2      //Falls start nicht null,
3      //dann gibt es was zu tun!
4      if (start != null) {
5          //Ausgabe des Namens, wenn älter als age
6          if (start.alter > age) {
7              System.out.println(start.name);
8          }
9          //rekursiver Aufruf mit nächstem Element der Liste
10         printNames_recursive(start.next, age);
11     }
12 }
13
14 static void printNames_iterativ(PersonItem start, int age) {
15     for(PersonItem item = start; item != null; item = item.next){
16         if (item.alter > age) {
17             System.out.println(item.name);
18         }
19     }
20 }
```

5 Hörsaalverwaltung (20 Pkte.)

Die Hörsaalverwaltung der TUD benötigt eine neue Software. Sie gehören einem Team an, dass diese Software entwickelt. Heute erhalten Sie die Aufgabe eine Java-Klasse zu entwerfen, die einen Hörsaal mit seinen Eigenschaften beschreibt.

Folgende Eigenschaften sollte ihre Klasse, der Sie den Namen `LectureRoom` geben, unterstützen:

- *Gebäudenummer* (building)
- *Raumnummer* (roomNumber)
- *Anzahl der Sitzplätze* (seats)

Diese Eigenschaften sind für jedes Objekt unveränderlich, d.h. sie sollen nur bei der Erzeugung eines Hörsaal-Objektes mit Werten belegt werden können.

Folgende Funktionen sollten Sie berücksichtigen und implementieren:

1. Entwerfen Sie geeignete Datenkomponenten für diese Klasse. Denken Sie daran, welche Zugriffsrechte eventuell benötigte Variablen haben sollten.
2. Schreiben Sie einen Konstruktur für diese Klasse.
3. Schreiben Sie Methoden, die die Werte der Eigenschaften zurückgeben (z.B. `getRoomNumber()`).
4. Schreiben Sie eine Methode `toString()`, die bei Aufruf Text folgender Art in Abhängigkeit der tatsächlichen Hörsaaldaten erzeugt und zurückgibt: `Hörsaal: S202/C205 (256 Plaetze)`

Lösungsvorschlag:

```
1  public class LectureRoom {
2      private String building;
3      private String roomNumber;
4      private int seats;
5
6      public LectureRoom(String building, String roomNumber, int seats)
         {
7          this.building = building;
8          this.roomNumber = roomNumber;
9          this.seats = seats;
10     }
11
12     public String getBuilding() {
13         return building;
14     }
15
16     public String getRoomNumber() {
17         return roomNumber;
18     }
19
20     public int getSeats() {
21         return seats;
22     }
23
```

```
24      public String toString() {
25          return "Hörsaal: " + building + "/" + room + " (" + seats + "
              Plaetze)";
26      }
27  }
```

6 Hörsaalbelegung (30 Pkte.)

Die in der vorhergehenden Aufgabe vorgestellte Hörsaalverwaltung soll um eine Belegungsverwaltung erweitert werden. Die Belegungsverwaltung soll prüfen, wenn ein Hörsaal für ein vorgegebenes Datum reserviert wird, dass dieser Saal auch tatsächlich zu diesem Termin verfügbar ist. Falls nicht, soll eine entsprechende Fehlermeldung den Belegungskonflikt mitteilen.

Erben Sie von der Klasse `LectureRoom`, die in der vorhergehenden Aufgabe erstellt wurde. Unabhängig von Ihrer Lösung in der vorhergehenden können Sie davon ausgehen, dass die Klasse `LectureRoom` folgende Methoden und Konstruktoren besitzt.

- **public** LectureRoom(String building, String roomNumber, **int** seats) – Initialisiert ein neu erzeugtes Objekt mit der Gebäude- und Raumnummer und der verfügbaren Anzahl der Sitzplätze.
- **public** String toString() – gibt einen String in der Form `Hörsaal: S202/C205 (256 Plaetze)` zurück.

Schreiben Sie nun eine neue Klasse `LectureRoomAllocation` die von der Klasse `LectureRoom` erbt und implementieren Sie folgende Methoden:

- **public void** doReservation(String date) – Diese Methode bekommt ein Datum als `String` übergeben (z.B. `20.10.2005`). In der Methode prüfen Sie, ob dieses Datum schon belegt ist. Falls ja, dann lösen Sie eine Ausnahme vom Typ `Exception` mit einer aussagekräftigen Fehlermeldung aus. Falls nein, dann tragen Sie dieses Datum als belegt ein.
- **public** String toString() – In dieser Methode erzeugen Sie einen String in der Form: `Hörsaal: S202/C205 (256 Plaetze) ist belegt am 06.10.2005 27.10.2005`. Sollte keine Belegung existieren, dann erzeugen Sie einen String in der Form: `Hörsaal: S202/C205 (256 Plaetze) ist nicht belegt`. Verwenden Sie sinnvoll auch die überschriebene `toString()`-Methode der Superklasse.
- Sie müssen auch einen Konstruktor definieren, der alle notwendigen Initialisierungswerte übergeben bekommt und diese an den Konstruktor der Superklasse weiterleitet.

Hinweis: Es ist hilfreich, die Klasse `HashSet` aus dem Package `java.util` zu verwenden. Falls Sie diese Klasse verwenden, vergessen Sie nicht, sie zu importieren. Deren Methode `add` liefert `true` zurück, falls das übergebene Element eingefügt werden konnte, ansonsten `false`, weil das Element schon vorhanden ist. Die Methode `iterator` liefert ein Iterator-Objekt zurück. Die Methode `isEmpty`, liefert `true`, falls kein Element in der Menge ist.

Lösungsvorschlag:

```
1  import java.util.HashSet;
2
3  public class LectureRoomAllocation extends LectureRoom {
4
5     Set allocations = new HashSet();
6
7     public LectureRoomAllocation (String building, String roomNumber,
        int seats) {
8        super(building, roomNumber, seats);
9        //eventuell auch:
10       //allocations = new HashSet();
11    }
```

```
12
13      public void doReservation(String date) throws Exception {
14          if (!allocations.add(date)) {
15              //Falls der Einfügebefehl nicht erfolgreich
16              //ausgeführt werden kann, dann Exception.
17              throw new Exception("Der Hörsaal ist für diesen Tag bereits
                    belegt: " + date));
18          }
19      }
20
21      public String toString() {
22          String returnString = super.toString();
23
24          if (allocations.isEmpty()) {
25              returnString = returnString + " ist nicht belegt.";
26          } else {
27              returnString = returnString + " ist belegt am";
28
29              Iterator it = allocations.iterator();
30
31              //Iteriere über die vorhandenen Datumseinträge
32              while (it.hasNext()) {
33                  returnString = returnString + " " + it.next();
34              }
35          }
36          return returnString;
37      }
38  }
```