Proseminar Programmiermethodik

Universität Innsbruck - Institut für Informatik

A. Blaas, M. Mayerl, B. Murauer, M. Tschuggnall

Blatt 2: Objektorientiertes Programmieren

Diskussionsteil

a) ★ Gegeben seien die folgenden Klassen:

```
1
2     public class Car {
3       private int carId;
4       private String brand;
5       private String model;
6     }
7     public class Company {
8       private int companyId;
9       public String name;
10
11      public Company(){
12        System.out.println("Created a new Company object");
13      }
14    }
15
16    public class Ferry {
17      private int ferryNumber;
18      private int capacity;
19
20      public Ferry(int ferryNumber, int theCapacity) {
21        this.ferryNumber = ferryNumber;
22        capacity = theCapacity;
23      }
24    }
```

- Welche Bedeutung hat das Wort `this`? Was würde geschehen, wenn man im Konstruktor der `Ferry`-Klasse dieses Schlüsselwort weglassen würde? Warum funktioniert es mit dem Attribut `capacity`?
- Schauen Sie sich folgenden Code an. Welche Statements funktionieren, und welche nicht? Warum?

```
1     public static void main(String[] args){
```

```
2        Car car = new Car();
3        Company company = new Company();
4        company.companyId = 13;
5        company.name = "MyCarCompany";
6        Ferry ferry = new Ferry();
7      }
```

b) ★ Erstellen Sie eine Klasse `Island` (Insel) und speichern Sie folgende Eigenschaften:

- Name
- Größe in km^2
- Anzahl der Einwohner

Erstellen Sie folgende Konstruktoren und Methoden:

- einen leeren Konstruktor, der keine Argumente entgegennimmt.
- einen Konstruktor, der Name und Größe entgegennimmt.
- einen Konstruktor, der Name und Größe und Anzahl der Einwohner entgegennimmt. Verwenden Sie dazu den vorigen Konstruktor.
- eine Methode, die alle vorhandenen Infos in einem String zusammenfasst und diesen zurückgibt.
- Erstellen Sie außerdem für jedes Attribut Getter- und Setter-Methoden.
- eine `main`-Methode, die drei Inseln anlegt und diese ausgibt. Verwenden Sie dazu jeweils einmal jeden der oben definierten Konstruktoren.

c) ★★ Die Klassen aus den vorigen Aufgaben sollen nun erweitert werden. Dabei soll ein Teil davon mit Java-Listen umgesetzt werden - deshalb hier eine kleine Einführung:

Um mehrere Elemente in einer Variable zu speichern, gibt es verschiedene Möglichkeiten. Wie in anderen Programmiersprachen kann man auch in Java Arrays verwenden. Durch die umständliche Handhabung der Indizes werden diese in der Praxis aber meistens nur für rechenintensive Prozesse verwendet. Für andere Aufgabengebiete gibt es die komfortableren `Collections`. Die genauen Hintergründe dieser Klassen erfordern allerdings Wissen aus *Generische Programmierung* und *Vererbung*, die erst später in der Vorlesung erklärt werden. Für die kommenden Beispiele reicht es aus, wenn Sie die Klassen so verwenden, wie sie unten beschrieben werden. Folgendes Beispiel beinhaltet die grundlegenden Konzepte von Listen, die anschließend erklärt werden.

```
1   import java.util.List;
2   import java.util.ArrayList;
3
4   public class TestLists {
5     public static void main(String[] args){
6       List<String> stringList = new ArrayList<>();
7       stringList.add("One");
8       stringList.add("Two");
9           // should print "Two", as counting starts at 0
10      System.out.println(stringList.get(1));
11      for (String s : stringList){
```

```
12          System.out.println(s);
13        }
14        stringList.remove(0);
15        System.out.println(stringList.size()); // should print "1"
16      }
17    }
```

In Zeilen 1 + 2 erfolgen *Imports*. Das Interface `List` ist von Java bereits vorgegeben, muss aber von einer anderen Klasse importiert werden, um sie zu verwenden. Dieses Statement muss noch vor der Klassendefinition erfolgen. IDEs wie Eclipse oder IntelliJ können diese Importe auch automatisch auflösen (In Eclipse: ctrl + ⇧ + O). Beachten Sie dabei, dass es womöglich noch andere Klassen mit dem Namen `List` gibt (z.B. `java.awt.List`) und importieren Sie die richtige (`java.util.List`).

In Zeile 6 wird eine Liste instantiiert. Das `<String>` direkt nach dem `List`-Objekttyp gibt an, dass in dieser Liste nur Objekte der Klasse `String` gespeichert werden können. Dieses Konzept wird im Thema *Generische Programmierung* behandelt.

Da `List` in Java ein Interface, und keine Klasse ist, müssen Sie einen konkreten Subtypen davon instantiieren (z.B. `ArrayList`). Der gewählte Subtyp bestimmt, welche spezielle Implementierung dann für die Liste verwendet wird (z.B. basiert `ArrayList` auf klassischen Arrays, während `LinkedList` als verkettete Liste implementiert ist). Details zu Interfaces und Klassen werden im Thema *Vererbung* genauer besprochen.

In den Zeilen 7 + 8 werden Einträge mit der Methode `add` zur Liste hinzugefügt. Die Reihenfolge der Einträge bleibt dabei erhalten: das Erste Element, das zur Liste hinzugefügt wird, erhält den Index 0, das zweite 1 usw.

Um ein Element aus der Liste zu holen, wird die Methode `get` verwendet, die den Index des gewünschten Eintrags entgegennimmt. In Zeile 9 wird dabei das zweite Element aus der Liste gelesen, da die Indizes bei 0 anfangen.

Um mit allen Elementen einer Liste bestimmte Operationen durchzuführen, wird eine foreach-Schleife verwendet. In den Zeilen 10 - 12 wird dies verwendet, um die Einträge der Liste auszugeben.

In Zeile 13 wird ein Element aus der Liste entfernt. Dabei kann man entweder den zu löschenden Index oder den zu löschenden Eintrag angeben. Eine Liste kann durch eine Löschoperation keine Lücken erhalten. Wird ein Element gelöscht, so werden die nächsten Elemente nachgerückt.

Die Menge der bereits eingetragenen Elemente kann mit `size()` ermittelt werden und wird in Zeile 14 verwendet.

Erweitern Sie nun die bestehenden Klassen wie folgt:

- Speichern Sie beim Auto zusätzlich die Firma, die es gebaut hat.
- Speichern Sie bei der Fähre eine Liste von Autos, die damit transportiert werden.
- Speichern Sie für jede Insel alle Fähren, die sie angefahren haben.
- Schreiben Sie eine `main`-Methode und erzeugen Sie Beispiel-Autos, Firmen, Fähren und verknüpfen Sie diese miteinander. Erzeugen Sie entsprechende Getter/Setter, falls Sie

diese benötigen.

Übungsteil (selbstständig zu lösen)

Aufgabe 1 (Kursverwaltung) [6 Punkte]

In diesem Beispiel sollen die Grundlagen objektorientierter Programmierung anhand eines fiktiven Kursverwaltungssystems geübt werden. Dazu werden die Klassen `Student`, `Group` und `GroupManagement` erstellt, die zur Verwaltung genau eines Proseminars verwendet werden können.

Erstellen Sie nun alle notwendigen Klassen, Methoden und Attribute nach den folgenden Vorgaben. Überlegen Sie sich jeweils, ob letztere `public` oder `private` definiert werden sollen:

a) *1 Punkt* Erstellen Sie eine Klasse `Student` mit den Attributen `id` (int), `firstname` und `lastname`. Erzeugen Sie 3 Konstruktoren:

- einen leeren Konstruktor mit keinen Argumenten
- einen Konstruktor, der die `id` des/der Studenten/in übernimmt
- einen Konstruktor, der die `id`, den Vor- und den Nachnamen übernimmt

Definieren Sie Getter und Setter - Methoden für alle Attribute.

b) *1 Punkt* Erstellen Sie eine Klasse `Group` und verwalten Sie darin die Gruppennummer, den/die GruppenleiterIn, die maximale Anzahl an StudentInnen in der Gruppe sowie eine Liste, die die in der Gruppe teilnehmenden StudentInnen speichert. Definieren Sie einen Konstruktor, der die Gruppennummer und die maximale TeilnehmerInnenzahl entgegennimmt.

Definieren Sie nun eine Methode `addStudent(Student student)`, die den übergebenen `Student` zur Gruppe hinzufügt. Kann der/die übergebene TeilnehmerIn nicht mehr in die Gruppe aufgenommen werden, da die maximale Anzahl bereits erreicht ist, so geben Sie `false` zurück, andernfalls `true`.

Definieren Sie eine Methode `printStudents`, die die Gruppe am Bildschirm nach folgendem Muster ausgibt:

`Group` *Gruppennummer* (*GruppenleiterIn*) *Vorname1 Nachname1*
`Group` *Gruppennummer* (*GruppenleiterIn*) *Vorname2 Nachname2*
...

Erzeugen Sie schlussendlich noch Getter und Setter - Methoden. Überlegen Sie sich dabei, ob und für welche Attribute Sie jeweils Getter und/oder Setter bereitstellen wollen.

c) *0.5 Punkte* Möchte man von einem/einer gegebenen Studenten/in die zugehörige Gruppe wissen, so müsste man aktuell von allen Gruppen alle Student-Listen durchsuchen. Um dies effizienter zu gestalten, soll in der Klasse `Student` zusätzlich die Gruppe als Attribut gespeichert werden. Fügen Sie dieses der Klasse `Student` hinzu und definieren Sie dafür Getter und Setter.

In der nun zu erstellenden Klasse `GroupManagement` soll schlussendlich alles verbunden werden. Übernehmen Sie dabei den Rumpf wie in der Datei `GroupManagement.java` (OLAT) angegeben.

d) *1 Punkt* Verwenden Sie die mitgelieferte Klasse `RandomNameGenerator.java` (OLAT), um zufällige Vor- und Nachnamen zu erzeugen. Passen Sie dazu das `package` Ihrem Projekt an[1].

[1] Schauen Sie sich dazu z.B. die Klasse `Student` an: falls ganz oben eine package-Deklaration steht bzw. von Eclipse erzeugt wurde, dann können Sie diese kopieren und im `RandomNameGenerator` ersetzen. Wenn kein package definiert ist, dann löschen Sie einfach die package-Zeile im `RandomNameGenerator`.

e) *1 Punkt* Implementieren Sie die Methode `createGroups()`: Erstellen Sie die vordefinierte Anzahl (`numberOfGroups`) an Gruppen mit der maximalen TeilnehmerInnenzahl (`maxNumberOfStudentsPerGroup`) und erzeugen Sie zufällig GruppenleiterInnen.

f) *1 Punkt* Implementieren Sie die Methode `createAndAddStudents()`: Erzeugen Sie die vordefinierte Anzahl an StudentInnen (`totalNumberOfStudents`) mit zufälligen Namen und aufsteigenden Id's, beginnend mit 1000. Verwenden Sie die in der Klasse `Group` definierte Methode `addStudent`, um die Studenten/innen der Gruppe hinzuzufügen. Fügen Sie in jede Gruppe die maximale Zahl an Studenten/innen ein.

g) *0.5 Punkte* Führen Sie nun die `main` - Methode von `GroupManagement` aus. Wenn Sie alles korrekt umgesetzt haben, wird am Bildschirm eine Liste der Gruppen mit den zugehörigen TeilnehmerInnen ausgegeben.

Aufgabe 2 (Erweiterte Kursverwaltung) [4 Punkte]

Ein Student (Nr. 87 in der Liste) meldet sich mit einem triftigen Grund[2] und möchte die Gruppe wechseln. Ein Java-Neuling nimmt sich der Aufgabe an und implementiert folgende Methode in der Klasse `GroupManagement`:

```
1  public void moveStudent(Student student, int newGroupNumber) {
2   student.getGroup().setGroupNumber(newGroupNumber);
3  }
```

a) *0.5 Punkte* Wo liegt das Problem bei dieser Implementierung? Fügen Sie zum Veranschaulichen des Problems obige Methode ein und führen Sie die angepasste `main`-Methode aus:

```
1  public static void main(String[] args) {
2   GroupManagement management = new GroupManagement();
3   management.createGroups();
4   management.createAndAddStudents();
5   System.out.println("\n\n-- INITIAL GROUPS --\n\n");
6   management.printGroups();
7
8   Student toMove = management.getStudents().get(87);
9   System.out.println("\n\nMoving student " + toMove.getFirstname() + " " +
10   + toMove.getLastname() + " to group 1");
11
12  management.moveStudent(toMove, 1);
13  System.out.println("\n\n-- GROUPS AFTER MOVING STUDENT --\n\n");
14  management.printGroups();
15 }
```

Im Folgenden soll nun schrittweise eine korrekte Implementierung zur Verschiebung von StudentInnen erstellt werden:

[2] am Vortag des PS-Termins nimmt der Student regelmäßig an einer WG-Feier teil, weshalb die aktuelle Gruppe (Beginn: 12 Uhr) nicht tragbar ist. Außerdem besteht eine Fahrzeit von über 7 min zur Universität, bei starkem Verkehr sogar über 10 min... – ein Gruppenwechsel ist also dringend nötig, um den rechtzeitigen Abschluss des Studiums nicht zu gefährden.

b) *1 Punkt* Fügen Sie der Klasse `Group` eine Methode `getStudentPositionInList(Student student)` hinzu, welche die Position des gegebenen Studierenden in der Liste zurückgibt (oder `-1`, falls nicht gefunden). Um die Position zu ermitteln, müssen StudentInnen auf Gleichheit überprüft werden. Da dies mit einem Vergleich wie `if(student.get(i) == student)` nicht funktioniert (Sie sollten bereits wissen, warum), implementieren Sie eine neue Methode `boolean equals(Student student)` in der Klasse `Student`, die zurückgibt, ob der/die gegebene StudentIn mit dem aktuellen Objekt übereinstimmt. Prüfen Sie dazu ausschließlich die `id`'s auf Gleichheit.[3]

c) *0.5 Punkte* Implementieren Sie eine Methode `boolean removeStudent(Student student)` in der Klasse `Group`, die eine/n gegebenen TeilnehmerIn aus der Gruppe entfernt. Verwenden Sie dazu die Methode aus der vorigen Teilaufgabe, um die Position zu finden. Geben Sie `true` zurück, wenn der/die StudentIn gefunden und entfernt wurden, andernfalls `false`.

d) *1.5 Punkte* Korrigieren/Erweitern Sie schlussendlich die Methode `moveStudent()` in der Klasse `GroupManagement`.

- Entfernen Sie den/die gegebene/n Studenten/in aus der aktuellen Gruppe.
- Versuchen Sie, den/die Studenten/in in die neue Gruppe einzufügen. Falls dies nicht möglich ist, da die neue Gruppe bereits voll ist: Wählen Sie zufällig eine/n Studenten/in aus dieser Gruppe und verschieben Sie diese/n in die Gruppe, in welcher der/die ursprünglich zu verschiebende StudentIn war (die beiden Gruppen werden also getauscht). Zum Erzeugen einer Zufallszahl im Bereich `[0;n]` können Sie die Util-Klasse `Random` verwenden:

```
1    int randomNumber = new java.util.Random().nextInt(n+1);
```

- Vergessen Sie nicht, die neue Gruppe auch in der Klasse `Student` zu speichern.

e) *0.5 Punkte* Testen Sie Ihre Implementierung, indem Sie die angepasste `main`-Methode ausführen und den Output kontrollieren.

Wichtig: Laden Sie bitte Ihre Lösung (.txt, .java oder .pdf) in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.

[3] Sie werden in der Vorlesung später bessere Techniken kennenlernen, um die Überprüfung auf Gleichheit umzusetzen, die "Listen-kompatibel" sind.