• Abgabe: Quellcode zu 5-1 auf Papier und ILIAS, Lösung von 5-2 auf Papier.

Aufgabe 5-1

Laden Sie von ILIAS die Dateien Address.java, FileTest.java, addresses.csv und addresses.txt herunter. Ihre Aufgaben sind:

Abgabetermin: **9.12.2011**

- 1. Schreiben Sie eine simple Unterklasse AddressFileException von Exception mit einem Konstruktor AddressFileException(String message), der nichts anderes tut als den Konstruktor der Superklasse aufzurufen.
- 2. Schreiben Sie eine Klasse AddressFile, die das Speichern und Laden von Address-Objekten in komma-getrennten Textdateien im folgenden Format ermöglicht:

```
1 , Max Frisch, Bahnhofstrasse 33 , 1001 , Zurich 2, Sophie Muster, Hauptstrasse 29, 1961, Fantasia
```

AddressFile benötigt einen Konstruktor, der als Argument einen Dateinamen erhält und diesen in einer Instanzvariable filename speichert. Implementieren Sie folgende Methoden (definieren Sie geeignete visibility modifiers!):

- (a) String toLine(Address addr): Wandelt ein Objekt adr in einen komma-getrennten String um.
- (b) Address parseLine(String line): Wandelt einen komma-getrennten String in ein Address-Objekt um. Wenn der String ein ungültiges Format hat, soll eine Exception vom Typ AddressFileException geworfen werden.
 - **Tipp**: Verwenden Sie die Scanner-Klasse mit ", " als Trennzeichen (delimiter) und lesen Sie die Werte jeweils mit next() aus. Achten Sie darauf, dass Leerzeichen vor und nach dem Komma ignoriert werden (siehe trim()-Methode der String-Klasse).
- (c) void save(ArrayList<Address> addresses): Speichert die in addresses enthaltenen Objekte mit Hilfe von toLine in der Datei filename ab.
- (d) ArrayList<Address> load(): Liest die Datei filename Zeile für Zeile ein und wandelt jede Zeile mit Hilfe von parseLine in ein Address-Objekt um. Die Gesamtheit der so erzeugten Objekte wird als ArrayList zurückgegeben.

Testen Sie AddressFile anhand der Klasse FileTest (zweiten Teil auskommentieren!).

3. Schreiben Sie nun eine zweite Klasse AddressFileLabelled, die das Lesen und Speichern von Address-Objekten mittels "beschrifteten" Dateien erlaubt:

```
id:1;name: Max Frisch; street: Gehweg 3; zip: 3001;city: Bern;
id:2; name:Anna Kunz;street: Hauptstr. 29; zip: 3001;city: Bern;
```

Nützen Sie das Konzept der Vererbung indem Sie die Klasse als Unterklasse von AddressFile definieren. Überschreiben Sie dabei **ausschliesslich** die Methoden toLine und parseLine.

Label/Wert-Paare lassen sich elegant mit Hilfe sogenannter regulärer Ausdrücke auslesen. Der folgende Codeausschnitt gibt "Max Frisch" zurück:

```
String line = "id: 3; name : Max Frisch; street: blabla;";
String label = "author";
Scanner scan = new Scanner(line);
scan.findInLine(label+"[\\s]*:[\\s]*([^;]*)");
return scan.match().group(1).trim();
```

4. Testen Sie beide Klassen mit FileTest und zeichnen Sie ein UML-Klassendiagramm aller involvierten Klassen (ohne die Java API-Klassen).

Abgabetermin: **9.12.2011**

Hinweis: Lesen/Schreiben von Dateien funktioniert in Java wie folgt (java.io.* importieren!):

Aufgabe 5-2

1. Welches ist die Ausgabe des folgenden Programms?

```
class A{
      protected String bla;
      public A(){ this.bla = "Hello from A"; }
      public void bla(){ System.out.println(this.bla); }
      public void foo(){ System.out.println("A.foo"); }
      public void bar(){ this.foo(); }
  class B extends A{
      public B(){
10
           super();
           this.bla = "Hello from B";
11
12
13
  class C extends B{
14
      public C(){ super(); }
15
      public void foo(){ System.out.println("C.foo"); }
16
  }
17
  new A().bla();
19
  new A().foo();
 new A().bar();
 new B().bla();
 new B().foo();
 new B().bar();
  new C().bla();
  new C().foo();
  new C().bar();
```

Wie verhält sich das Programm wenn man Zeile 16 durch folgende Methode ersetzt?

```
public void foo(){ this.bar(); }
```