

- **Abgabe:** Quellcodes zu 6-1 auf Papier und ILIAS, Lösungen zu 6-2 auf Papier. Vorgegebene Dateien müssen Sie **nicht** ausdrucken!

Aufgabe 6-1

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

1. 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, 1000, 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 adr)`: 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.
Tip: 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!).

2. 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 = "name";
Scanner scan = new Scanner(line);
scan.findInLine(label+"[\\s]*:[\\s]*([^;]*)");
return scan.match().group(1).trim();
```

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

Hinweis: Lesen/Schreiben von Dateien funktioniert in Java wie im folgenden Beispiel:

```
import java.io.*
...
Scanner fileScan = new Scanner(new File("in.txt"));
while (fileScan.hasNextLine()){
    line = fileScan.nextLine();
    ...
}

PrintWriter file = new PrintWriter(
    new BufferedWriter(new FileWriter("out.txt")));
file.println("bla");
file.close();
```

Aufgabe 6-2

Welches ist die Ausgabe des folgenden Programms? Überlegen Sie sich dies, ohne das Program auszuführen.

```
1 public class A{
2     protected String bla;
3     public A(){ this.bla = "Hello from A"; }
4     public void bla(){ System.out.println(this.bla); }
5     public void foo(){ System.out.println("A.foo"); }
6     public void bar(){ this.foo(); }
7 }
8 public class B extends A{
9     public B(){
10         super();
11         this.bla = "Hello from B";
12     }
13 }
14 public class C extends B{
15     public C(){ super(); }
16     public void foo(){ System.out.println("C.foo"); }
17 }
18 public class Test{
19     public static void main(String [] args) {
20         new A().bla();
21         new A().foo();
22         new A().bar();
23         new B().bla();
24         new B().foo();
25         new B().bar();
26         new C().bla();
27         new C().foo();
28         new C().bar();
29     }
30 }
```