Anmerkungen

- Dies ist eine Bonus-Übungsserie. Mit dieser Übung können Sie **einen** zusätzlichen Punkt für Ihr Testat erreichen.
- Der Punkt wird erreicht, wenn **zwei der drei** Aufgaben gut gelöst werden.
- Trotz der Bezeichnung *Bonus*, ist auch diese Übung klausurrelevant.
- Ansonsten gelten die gleichen Regeln wie bisher: Abgabe auf Ilias, Quellcode-Dateien müssen kompilieren und die Namen der Gruppenmitglieder müssen vermerkt sein.

Aufgabe 8-1

Programmieren Sie eine Klasse `Dictionary`, mit der man einem Wort eine Liste von Übersetzungen zuordnen kann. Laden Sie dazu die Klasse `Translator` (siehe ILIAS) herunter und implementieren Sie `Dictionary` so, dass `Translator` wunschgemäss funktioniert (natürlich ohne `Translator` zu ändern):

```
$ java Translator
Enter a word to translate: (ctrl+c to abort): Himmel
sky
heaven (rel.)

Enter a word to translate: (ctrl+c to abort): gugus
Word not found.
```

Achten Sie auf die Fehlerbehandlung. Falls für ein gesuchtes Wort keine Übersetzungen existieren, soll die Methode `translate` eine `WordNotFoundException` werfen.

Tipp: Verwenden Sie die Klasse `HashMap`. Die Klasse `Translator` dürfen Sie nicht abändern.

Aufgabe 8-2

Schreiben Sie eine Klasse `Queue`, die eine FIFO-Queue von beliebigen Objekten modelliert. `Queue` soll Methoden `enqueue`, `dequeue` und `isEmpty` haben, wobei `dequeue` eine `EmptyQueueException` (deren Klasse Sie schreiben müssen) wirft für den Fall dass die Queue leer ist. Testen Sie Ihre Klasse mit dem Programm `QueueTest` (siehe ILIAS):

```
$ java QueueTest
Queue is empty!
* print job of alice: Hi, I'm Alice.
* print job of bob: Hi, I'm Bob and I've been living next door to
Alice for 24 years.
* print job of anna: Hi, I'm Anna.
```

Hinweis: Klassen des Java Collections Framework dürfen Sie *nicht* verwenden. Die Klasse `QueueTest` dürfen Sie nicht abändern.

Aufgabe 8-3

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

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 = "name";
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).

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();
```