Adressbuch, 4. Ausbaustufe

Verwendet zusätzlich: Dateiein- und ausgabe, Serialisierung.

Kümmern wir uns nun mal um die Persistenz, also darum, dass unser Adressbuch einen Programmlauf überdauert und beim nächsten Mal wieder zur Verfügung steht. Diese Funktionalität soll zunächst einmal nicht in eine konkrete AddressBookDataStore-Klasse eingebaut werden, sondern in eigene Klassen ausgelagert werden. Auch wird es dabei erst einmal ganz primitiv zugehen, wir werden nämlich immer das komplette Adressbuch laden und speichern.

Um flexibel herumexperimentieren zu können, gebe ich wieder ein Interface und eine allgemeine Exception vor:

```
-----------------------------------------------------------------
package de.mpaap.addressbook;

public interface FullPersistence {
   AddressBookDataStore loadBook(String name) throws AddressBookException;

   void storeBook(AddressBookDataStore book, String name) throws AddressBookException;
}
-----------------------------------------------------------------
package de.mpaap.addressbook;

public class AddressBookException extends Exception {
   public AddressBookException() {

   }

   AddressBookException(String message) {
      super(message);
   }
}
-----------------------------------------------------------------
```

Der der Methode storeBook() übergebene Name dient dazu, verschiedene Adressbücher auseinanderhalten zu können. Was eine konkrete Implementierung von FullPersistence damit anfängt, werden wir nicht weiter festlegen. Ebenso lassen wir völlig offen, wie und wo eine konkrete Implementierung die Adressbuch-Daten ablegt. Ein konkretes FullPersistence-Exemplar kann diese Information direkt im Code festlegen, sie bei der Erzeugung mitbekommen oder irgendwoanders her beziehen.

Aufgaben:

1. Schreiben sie eine Klasse SerializationToFilePersistence, die das Interface FullPersistence implementiert und komplette Adressbücher im Dateisystem ablegt, indem es sie *serialisiert* bzw. *deserialisiert*. Verwenden Sie dazu die Methoden der Klassen ObjectOutputStream bzw. ObjectInputStream. Verwenden Sie als Ablageort für Adressbücher einen Ordner addressbooks im Heimatverzeichnis des Benutzers. Wo sich dieses befindet, erfahren Sie durch Abfrage von System.getProperty("user.home"). Fangen Sie alle zu erwartenden Exceptions ab, und erzeugen Sie im Catch-Block eine neue AddressBookException, der sie die Nachricht der „gefangenen" Exception im Konstruktor mitgeben. Testen Sie ihre Klasse, indem Sie ein ArrayListAddressBookDataStore aus Adressbuch_3 speichern und aus einem zweiten Testprogramm heraus laden und ausgeben. Welche Änderungen an Ihren Klassen aus Adressbuch_3 müssen Sie dazu vornehmen?

2. Nachdem Sie ein Adressbuch gespeichert haben, ändern Sie bitte die Klasse Person so ab, dass sie ein zusätzliches Attribut phone1 zur Aufnahme einer Telefonnummer erhält. Schreiben Sie zu die-

sem Attribut einer Getter- und eine Setter-Methode und passen sie die Methode toString() so an, dass auch dieses Attribut mit ausgegeben wird. Versuchen Sie nun, das gespeicherte Adressbuch zu laden. Versuchen Sie, mit Hilfe des Kapitels "Serialisierung" des "Handbuchs der Java-Programmierung" (kostenloser Download unter http://www.javabuch.de, online z.B. unter https://dbs.cs.uni-duesseldorf.de/lehre/docs/java/javabuch/html/k100257.html) zu verstehen, was hier geschieht und warum.

3. Machen sie die Änderungen aus 2. an der Klasse Person rückgängig. Versuchen Sie nun, mit Hilfe des genannten Kapitels, ihre Klasse Person so zu ändern, dass Adressbücher, die sie ab jetzt speichern, auch noch geladen werden können, wenn Sie später weitere Attribute hinzufügen. Testen Sie ihre Änderung, indem Sie ein Adressbuch speichern, und dann erneut das Attribut phone1, die Getter- und die Setter-Methode hinzufügen und die Methode toString() anpassen. Was fällt Ihnen bzgl. des Rückgabewertes von toString() auf, wenn Sie ein Adressbuch laden, das sie gespeichert haben, als es das Attribut phone1 noch nicht gab? Passen Sie die Methode so an, dass eine sinnvolle Ausgabe entsteht.