Schule Informationstechnik

der Bundeswehr

Sprachausbildung Java

Übungen

"Dateiverarbeitung"

Stand: Januar 2016

1 Verzeichnisbaum ausgeben I

a) Erstellen Sie ein Programm, welches die Verzeichnisstruktur (beachten Sie die Einrückung) und die Inhalte Ihres Nutzerverzeichnisses (*System.getProperty("user.dir");*) in folgender Form ausgibt:

\ <Name des Ordners>

\ <Name des Ordners>

- <Name der Datei>

\ <Name des Ordners>

- <Name der Datei>

- ...

- <Name der Datei>

- ...

- <Name der Datei>

- <Name der Datei>

- ...

Tipp: Nutzen Sie einen rekursiven Ansatz.

b) Modifizieren Sie das Programm im Hinblick auf eine sortierte Ausgabe der Inhalte. Hierbei sollen zunächst erst alle Dateien eines Ordners und dann erst die Unterordner (und bevor es mit dem [n+1]-ten Unterordner weitergeht, erst wieder die Dateien des [n]-ten Unterordner, etc.pp.) gelistet werden. Nutzen Sie in der Umsetzung die Schnittstelle `FileFilter`. Vorher wurden die Inhalte alphabetisch sortiert.

\ <Name des Ordners>

- <Name der Datei>

- ...

\ <Name des Ordners>

- <Name der Datei>

- <Name der Datei>

- ...

\ <Name des Ordners>

\ ...

2 Verzeichnisbaum ausgeben II

a) Ändern Sie Aufgabe 1 dahingehend ab, dass die Verzeichnisbäume nicht mehr gleich auf der Konsole ausgegeben, sondern zunächst in einer Datei gespeichert werden.
b) Erst nachdem der Verzeichnisbaum vollständig generiert wurde, soll dieser dann im zweiten Schritt aus der Datei gelesen und auf der Konsole ausgegeben werden.

3 Soldatenablage I

a) Erstellen Sie eine Klasse `Soldat`, welche die Attribute `Name`, `Vorname`, `PK`, `Dienstgrad` (wählen Sie geeignete Datentypen) besitzt und neben dem Standard-Konstruktor, Getter- und Setter-Methoden besitzt.
b) Erstellen Sie nun eine Klasse `Ablage`, welche eine Liste von Soldaten verwaltet. Ermöglichen sie das *Hinzufügen*, *Abfragen* und *Entfernen* von Soldaten. Der aktuelle Zustand der Ablage soll mit Hilfe der folgenden zwei Methoden und der Klasse `RandomAccessFile` gespeichert bzw. geladen werden:

```
/*
Speichert die aktuelle Liste von Soldat-Objekten nacheinander in die
Datei, welche im Konstruktor angegeben wurde. Hierbei wird die Datei
im Bedarfsfall erstellt, sollte sie bereits existieren, werden die
alten Daten überschrieben
*/
public void speichern() { … }
/*
Lädt die Daten der gespeicherten Soldat-Objekte aus der Datei und
stellt den gespeicherten Zustand der Ablage wieder her. Im Fehlerfall
ist die Liste der Ablage leer.
*/
public void laden() { … }
/* Z U S A T Z für schnelle Teilnehmer
Speichert die aktuelle Liste von Soldat-Objekten nacheinander in die
Datei, welche im Konstruktor angegeben wurde. Hierbei wird die Datei
im Bedarfsfall erstellt, sollte sie bereits existieren, werden die
Daten hinten angehängt.
*/
public void anhaengen() { … }
```

c) Erstellen Sie schließlich die Testklasse `TestAblage`, welche mindestens zwei `Soldat`-Objekte erstellt (wenn gewünscht, können Sie diese auch über die Konsole einlesen), sie danach per `Ablage.speichern()` als Datei ablegt und zum Schluss per `Ablage.laden()` wieder einlädt und dann auf der Konsole ausgibt.

4 Datei kopieren

Erstellen Sie eine Klasse *FileUtils*, welche die statische Methode *copyFile* (siehe unten) zur Verfügung stellt. Testen Sie ihre Klasse entsprechend in der zugehörigen *main*-Methode.

```
/*
Kopiert die in src angegebene Datei nach dest. Hierzu wird die
Basisfunktionalität der Klassen FileInputStream und FileOutputStream
genutzt.
*/
public static void copyFile( File src, File dst ) { … }
```

5 Soldatenablage II

Stellen Sie Aufgabe 3 (Soldatenablage I) von *RandomAccessFile* auf die Verwendung von *DataOutputStream* / *DataInputStream* um (bestehende Daten dürfen hier überschrieben werden).

6 Soldatenablage III

a) Stellen Sie Aufgabe 5 (Soldatenablage II) von `DataOutputStream / DataInputStream` auf die Verwendung von `ObjectOutputStream / ObjectInputStream` (unter Nutzung von `writeObject(Object o)` und `readObject()`) um.
b) Erweitern Sie die Umstellung dahingehend, dass nicht mehr die Bestandteile der Klasse `Soldat`, sondern die Klasse selbst (bzw. die komplette Liste) serialisiert werden.

7 (Zusatz-)Aufgabe 7

Erstellen Sie eine Java-basierte Shell, welche mindestens den folgenden Befehlssatz unterstützt: *cd*, *ls*, *mkdir*, *rmdir*, *touch*, *rm* und weiterhin eine Kommandozeile der Form <jAvA::<aktuelles Verzeichnis>> implementiert.

Tipp: `System.console()` kann u.U. nützlich sein (Voraussetzung ist allerdings, dass Sie das Programm von der Kommandozeile aus aufrufen).