**Proseminar Programmiermethodik**

Universität Innsbruck - Institut für Informatik

A. Blaas, M. Mayerl, B. Murauer, M. Tschuggnall

# Blatt 4: Collections

## Diskussionsteil

a) ★ Importieren Sie die folgenden Quellcode in die `main`-Methode einer Java-Klasse und machen sich mit den dort verwendeten `Java-Collections` vertraut:

```
1   String word0="auto";
2   String word1="zebra";
3   String word2="nashorn";
4   String word3="elefant";
5
6   Set<String> myFirstSet = new HashSet<>();
7   myFirstSet.add(word0);
8   myFirstSet.add(word1);
9   myFirstSet.add(word2);
10  myFirstSet.add(word3);
11  myFirstSet.add(word0);
12
13  List<String> myAlreadyKnownList = new ArrayList<>();
14  myAlreadyKnownList.add(word0);
15  myAlreadyKnownList.add(word1);
16  myAlreadyKnownList.add(word2);
17  myAlreadyKnownList.add(word3);
18  myAlreadyKnownList.add(word0);
```

Geben Sie die Elemente der beiden `Collections` am Bildschirm aus und erklären Sie, warum es zu dieser unterschiedlichen Ausgabe kommt. Beziehen Sie sich dabei auf die Charakteristika der zugrunde liegenden Datenstruktur.

Fügen Sie nun zusätzlich den folgenden Quellcode ein und erklären Sie wiederum die unterschiedliche Ausgabe.

```
1   Map<String,Integer> myFirstHashMap = new HashMap<>();
2   myFirstHashMap.put(word0, 10);
3   myFirstHashMap.put(word1, 29);
4   myFirstHashMap.put(word2, 1);
5   myFirstHashMap.put(word3, 100);
6
7   Map<String, Integer> myFirstTreeMap = new TreeMap<>();
```

```
8       myFirstTreeMap.put(word0, 10);
9       myFirstTreeMap.put(word1, 29);
10      myFirstTreeMap.put(word2, 1);
11      myFirstTreeMap.put(word3, 100);
```

Schlagen Sie außerdem nach, welche Charakteristiken die unterschiedlichen Datenstrukturen aufweisen, um für die nächste Teilaufgabe gewappnet zu sein.

b) ★ Geben Sie für die folgenden Szenarien jene Datenstruktur (Collection) an, die Sie am geeignetsten befinden und begründen Sie Ihre Wahl:

- Ein Sensor misst über mehrere Stunden hinweg sekündlich die Temperatur. Diese Daten sollen in einer Java-Collection zwischengespeichert (hinzugefügt) werden.
- Teilnehmer einer Lotterie werden in einer `Collection` abgespeichert und gut durchgemischt (geshuffelt). Es wird eine Nummer gezogen: Jener Kandidat, der sich an der Stelle in der `Collection` befindet, welche der gezogenen Nummer entspricht, gewinnt.
- Bei einem Verwaltungsprogramm, welches über 100.000 Einwohner einer Stadt in einer Collection verwaltet, soll der Einwohner mit der Sozialversicherungsnummer 'abc123' gefunden und zurückgegeben werden.
- Ein Bücherei speichert ihre Bücher in einer Collection ab und möchte diese absteigend nach ISBN anzeigen lassen (ohne einen Befehl zum Sortieren explizit aufrufen zu müssen).
- Bei einem Anti-Virus Programm werden die Dateinamen der bereits überprüften Dateien zwischengespeichert, um eine erneute Überprüfung zu vermeiden. Vor jeder (vermeintlich) neuen Datei soll sehr schnell überprüft werden, ob eine Datei mit diesem Namen bereits überprüft wurde.

c) ★★ Importieren Sie die Klasse `Person.java` (OLAT), welche Sie bereits in Blatt 3 verwendet haben und führen Sie folgende Anweisungen aus:

- Erstellen Sie eine Methode, welche eine bestimmte Anzahl (Parameter) an Personen anlegt und diese in einer List abspeichert. Verwenden Sie hier für den Generator für Zufallsnamen ( `RandomNameGenerator.java` (OLAT) ), welchen Sie bereits aus Blatt 2 kennen.
- Erstellen Sie nun eine Methode, welche eine Liste von Personen als Parameter erhält und diese aufsteigend nach Nachnamen sortiert (ohne eine neue `Collection` anzulegen). Überlegen Sie sich, welche Möglichkeiten sich Ihnen bieten, um diese Sortierung zu realisieren.
- Erstellen Sie eine weitere Methode, welche ebenfalls eine Liste von Personen erhält, diese aber absteigend nach Nachnamen sortiert (ohne eine neue `Collection` anzulegen). Beide Sortiermöglichkeiten sollen dabei gleichzeitig anwendbar sein, ohne dass Sie den Quellcode zwischenzeitlich ändern müssen!
- Überprüfen Sie, ob die Liste der Personen eine Person mit dem Namen 'Donald Duck' enthält. Verwenden Sie hier für die `contains(..)`-Methode, welche das List-Interface zur Verfügung stellt. Übergeben Sie der Methode eine neue Person mit dem Namen 'Donald Duck'. Ist das Ergebnis korrekt? Falls nein, wo könnte sich der Fehler befinden?

## Übungsteil (selbstständig zu lösen)

## Aufgabe 1 (Einfache Dateiverwaltung) [6 Punkte]

Erstellen Sie für diese Aufgabe einen einfachen Dateimanager, mit welchem bestimmte Operationen durchgeführt werden können. Machen Sie dabei Gebrauch von den `Java-Collections`. Wählen Sie jeweils jene `Collection` aus, welche Ihnen am effizientesten bzw. sinnvollsten erscheint. Begründen Sie kurz in einem Satz (Kommentar im Source-Code) ihre Wahl.

**Hinweis** ⚠

In den Unteraufgaben 1 und 2 steht der Begriff `Collection` stellvertretend für eine `Java-Collection` Ihrer Wahl. Sie müssen sich **NICHT** auf jene Datenstrukturen beschränken, welche das `Interface java.util.Collection` implementieren. D.h., Sie dürfen auch `Maps` verwenden. Beschränken Sie sich in Ihrer Umsetzung nicht ausschließlich auf `Lists`.

a) *0.5 Punkte* Erstellen Sie zunächst die Grundstruktur Ihres Dateimanagers. Dabei werden folgende Komponenten benötigt:

- Die Klasse `Author` enthält die Attribute Benutzername, Vorname sowie Nachname.
- Die Klasse `File` hat die Attribute Name, Author sowie Inhalt (Eine `Collection` von Strings). Der Inhalt eines `Files` besteht aus mehreren Wörtern, die in das `File` eingefügt werden können (Siehe code-snippet):

```
1    public void addContent(String word) {
2     ...
3    }
```

- Die Klasse `Folder` beinhaltet ebenfalls ein Attribut Name, Author sowie eine `Collection` von `Files`.

Erstellen Sie die notwendigen Klassen sowie `getter` und `setter` Methoden. Beachten Sie dabei die Namenskonventionen für Bezeichner in Java. Lassen Sie sich eine geeignete Vererbungshierarchie einfallen, damit Sie die Attribute, die bei mehreren Klassen vorhanden sind, nicht bei jeder Klasse erneut definieren müssen.

b) *0.5 Punkte* Fügen Sie nun den eben erstellten Klassen folgende Methoden hinzu:

- Für die Klasse `File`: Eine Methode, um Inhalt (Strings) zu einem `File` hinzuzufügen.
- Für die Klasse `Folder`: Eine Methode, um `Files` zu einem `Folder` hinzuzufügen.

Überschreiben Sie außerdem die entsprechenden `toString`-Methoden der Klassen `Author`, `File` und `Folder`, um eine angemessene Ausgabe durchführen zu können. Beachten Sie hierfür das folgende Beispiel (für `File`):

```
1    ...
2    System.out.println(file); //soll NICHT die Objekt-Referenz ausgeben
3    ...
```

c) *3 Punkte* Erstellen Sie nun ein `Interface FolderManager` mit den nachfolgenden Methoden (Sie können das `Interface` `FolderManager.java` (OLAT) kopieren). Halten Sie sich dabei strikt an die vorgegebene Signatur!

**Hinweis**

Anders als bei den vorherigen Unteraufgaben bezeichnet der Typ `Collection` das `Interface java.util.Collection`. Verwenden Sie `Iteratoren`, um Ihre Datenstrukturen zu durchlaufen.

```
1    public void addFolder(Folder folder);
2    public Collection<Folder> findFoldersByAuthor(Author author);
3    public Collection<Folder> getAndSortFoldersByName();
4    public Collection<Folder> getAndSortFoldersByAuthor();
5    public void printContent(Collection<Folder> folders, boolean foldersOnly);
```

Erstellen Sie nun eine Klasse `BasicFolderManager`, welche das zuvor erstellte `Interface` implementiert. Sehen Sie sich dazu die Dokumentation der Methoden in `FolderManager.java` (OLAT) an.

**Hinweis**

Schauen Sie sich die zu implementierenden Methoden genau an und überlegen Sie sich, welche `Collection` für eine bestimmte Methode den größten Vorteil bringt. Es empfiehlt sich daher, sich nicht auf eine einzelne `Collection` zu verlassen, in der Sie alle Daten speichern. Verwenden Sie mehrere `Collections`, die Sie beim Aufruf der Methode `addFolder(...)` befüllen können (um somit eine schnellstmögliche Ausführungszeit zu garantieren).

Beachten Sie, dass es außerdem auch möglich ist, `Collections` zu verschachteln:

```
1    // jedem key ist eine Liste zugeordnet
2    public Map<String, List<File>> nestedCollection = new HashMap<>();
```

d) *2 Punkte* Testen Sie Ihre Implementierung ausgiebig, indem Sie Testdaten erstellen und Ihre Implementierung anhand dieser Daten testen. Generieren Sie sich dafür `Folder`, welche `Files` enthalten.

## Aufgabe 2 (Fortgeschrittene Dateiverwaltung) [4 Punkte]

Basierend auf der vorherigen Aufgabe gilt es bei dieser Aufgabe, Ihren bestehenden Dateimanager zu erweitern: Während bislang fast ausschließlich Operationen auf `Folder` unterstützt werden, soll die neue Version Ihres Dateimanagers außerdem Operationen auf die in den `Foldern` enthaltenen `Files` anbieten.

a) *3 Punkte* Erstellen Sie ein weiteres `Interface FileManager` mit den folgenden Methoden (Sie können das `Interface FileManager.java` (OLAT) kopieren).

```
1    public Collection<File> findFilesContaining(String word);
2    public Collection<File> findFilesByAuthor(Author author);
3    public boolean fileExists(String folderName, String fileName);
4    public boolean fileContainsWord(File file, String word);
5    public void printFiles(Collection<File> files);
```

Erstellen Sie nun eine Klasse `FileAndFolderManager`, welche das zuvor definierte `Interface` (`FileManager`) implementiert sowie von `BasicFolderManager` erbt (da Sie ja die vorhandene Funktionalität Ihres `BasicFolderManager` behalten möchten). Sehen Sie sich dazu die Dokumentation der Methoden in `FileManager.java` (OLAT) an. Überschreiben Sie die notwendigen Methoden der Superklasse `BasicFolderManager`.

**Hinweis** ⚠

Für diese Aufgabe werden Sie weitere Datenstrukturen benötigen (z.B.: eine `Collection`, bei welcher Sie mitspeichern, welche `Files` ein bestimmtes Wort enthalten).
Dieses 'Mitspeichern' lösen Sie am besten, wenn Sie die Methode `addFolder(...)` der Superklasse überschreiben und somit bei jedem Hinzufügen eines neuen `Folders` Ihre gewählte Datenstruktur updaten. Achten Sie außerdem darauf, dass Ihre Datenstrukturen der Superklasse nach wie vor befüllt werden!

b) *1 Punkt* Erweitern Sie nun die Generierung der Testdaten, indem Sie jedem `File` Wörter (Strings) hinzufügen. Testen Sie nun Ihre Klasse `FileAndFolderManager` ausgiebig.

**Wichtig:** Laden Sie bitte Ihre Lösung (.txt, .java oder .pdf) in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.