Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Programming Concepts and Paradigms (PCP)

PCP-Übung zu Java 8, Teil 1 + 2 (Woche 8)

Hauptthemen: Default- und statische Interface Methoden, Lambda-Ausdrücke, funktionale Interfaces, Methoden-Referenzen, Streams, Aggregate-Operations (intermediate/terminal, stateless/stateful, short-circuiting), Lambda-Anwendungen

Ruedi Arnold

Diese Übung repetiert und vertieft den in Java 8, Teil 1 + 2 vermittelten Stoff. Gehen Sie zur Vorbereitung nochmals durch die Unterrichtsfolien durch und schauen Sie insbesondere, dass Sie alle auf den Folien angegebenen Code-Beispiele verstanden haben und nachvollziehen können.

1. Default-Methoden & Mehrfach Vererbung

In der Vorlesung haben Sie gesehen, dass Interfaces seit Java 8 auch Implementierungen von Methoden haben können in Form von default- und statischen Methoden. Gegeben sind die folgenden zwei Interfaces und die folgende Klasse:

```
public interface GeneralInterface {
    default public void doIt() {
        System.out.println("Do it the GENERAL way.");
    }
}

public interface SpecificInterface extends GeneralInterface {
    @Override
    default public void doIt() {
        System.out.println("Do it the SPECIFIC way.");
    }
}

public class Exercise1 implements GeneralInterface, SpecificInterface {
    public static void main(String[] args) {
        Exercise1 ex1 = new Exercise1();
        ex1.doIt();
    }
}
```

a) Was gibt das Programm aus, wenn die main-Methode der Klasse `Exercise1` aufgerufen wird?

b) Die beiden Interfaces `GeneralInterface` und `SpecificInterface` implementieren beide die Methode `public void doIt()` und die Klasse `Exercise1` implementiert beide Interfaces. Warum führt dies zu keinen Problemen mit Mehrfachvererbung?

c) Angenommen, beim `SpecificInterface` würde `extends GeneralInterace` gestrichen. Gäbe es dann ein Problem mit Mehrfachvererbung? Falls Ja: Wie könnte dieses behoben werden? Falls Nein: Begründen Sie Ihre Antwort.

2. Iterable.forEach: Zahlen und Quadratzahlen ausgeben

Implementieren Sie eine Methode, welche für einen Array von Ganzzahlen diese Zahlen sowie die dazugehörigen Quadratzahlen ausgibt. Hinweis: Ihre Methode soll dazu in sinnvoller Weise die forEach-Methode vom Interface Iterable verwenden, verwandeln Sie den Array also zuerst in eine Collection. Die Signatur der zu implementierenden Methode sieht wie folgt aus:

```
public void printNumbersAndSquares(Integer[] numbers)
```

Der Aufruf dieser Methode mit dem Array `{1, 2, 3, 5, 8}` als Argument soll beispielsweise folgende Ausgabe produzieren:

```
1: 1
2: 4
3: 9
5: 25
8: 64
```

3. Eigenes funktionales Interface: TripleIntOperator

Sie haben in der Vorlesung gesehen, dass wir selber eigene funktionale Interfaces implementieren können. Wir sprechen auch von SAM-Typen, für Single Abstract Method.

a) Programmieren Sie ein eigenes funktionales Interface mit dem Namen `TripleIntOperator`, welches drei `int`-Argumente nimmt.

b) Wie muss die entsprechende SAM-Methode von ihrem Interface heissen?

c) Von welchem Typ muss dabei die Rückgabe der abstrakten Methode gemäss Namenskonvention des funktionalen Interfaces sein?

d) Instanziieren Sie Ihren eigenen Operator mit einem Lambda-Ausdruck, welcher die Summe der drei Argumente zurück liefert. Vervollständigen Sie also folgende Zeile:

```
TripleIntOperator tio = ...
```

Testen Sie ihre Lambda-Implementierung, indem Sie die entsprechende Interface-Methode aufrufen, z.B. mit den Argumenten 1, 2 und 3.

4. Streams & Lambdas

Implementieren Sie eine Methode `public String processNames(String[] names)`, welche für einen Array von Strings alle Strings mit einer Länge zwischen 3 und 4 (je inklusiv) in Grossbuchstaben umwandelt und, mit einem Abstand dazwischen, zusammen hängt. Für den Array `{"Susanna", "Joe", "Lu", "Timmy", "Rafael", "Lisa"}` soll die Rückgabe also wie folgt aussehen: `"JOE LISA "`.
Hinweise: Verwenden Sie dazu einen String-Stream sowie passende Aggregate-Operationen. Verwenden Sie an passender Stelle eine Methoden-Referenz auf eine String-Methode.

5. Sequentielle und parallele Streams

Sie haben in der Vorlesung gesehen, dass es sequentielle und parallele Streams gibt.

a) Implementieren Sie eine Code-Sequenz, die alle Ganzzahlen von 0 bis 99 ausgibt. Verwenden Sie dazu einen `IntStream`, sowie dessen `iterate`- und `limit`-Methoden. Die Ausgabe sollte also wie folgt aussehen:
```
0, 1, 2, 3, 4, 5, 6, 7, ... 96, 97, 98, 99,
```

b) Modifizieren Sie Ihren Code, so dass neu ein paralleler Stream verwendet wird. Die Ausgabe sieht jetzt beispielsweise so aus wie unten im Beispiel. Warum sind die Zahlen jetzt nicht mehr geordnet? Erläutern Sie diesen Sachverhalt.

```
65, 56, 57, 71, 28, 31, 15, 43, 90, 44, 45, 16, 17, 12, 13, 14, 32, 29, 30, 72, 58, 59, 60, 66, 61, 73, 74, 25,
26, 27, 33, 21, 22, 48, 91, 92, 49, 23, 34, 81, 68, 69, 70, 53, 54, 55, 50, 51, 67, 52, 62, 63, 96, 97, 82, 83,
35, 36, 24, 46, 47, 87, 75, 76, 77, 18, 19, 78, 79, 80, 84, 98, 64, 40, 37, 41, 99, 85, 86, 9, 10, 11, 6, 20,
93, 94, 95, 88, 89, 7, 0, 1, 3, 4, 42, 38, 5, 2, 8, 39,
```

6. Unendlich viele (Pseudo-)Zufallszahlen

Sie haben in der Vorlesung gesehen, dass Streams grundsätzlich unendlich lange sein können.

a) Implementieren Sie eine Code-Sequenz, die unendlich viele zufällige Ganzzahlen erzeugt, und abbricht, wenn eine Zahl gefunden wurde, welche zwischen 10'000 und 12'000 liegt. Verwenden Sie für die Erzeugung vom entsprechenden `IntStream` die Methode `ints()` von `java.util.Random` und dann zum Filtern die Stream-Operation `anyMatch(...)`.

b) Erweitern Sie Ihren Code so, dass ausgegeben wird, an welcher Stelle im Stream eine passende Zahl gefunden wurde. Hinweis: Das ist nicht ganz trivial, da gemäss Scoping-Regeln von Lambdas sichtbare Variablen „effectively final“ sein müssen. Implementieren Sie eine einfache Lösung, welche nicht auf parallele Streams eingeht und entsprechend nicht Thread-safe ist. Die Ausgabe soll dann also beispielsweise wie folgt aussehen:

```
Found suitable number at stream position 2064645
```