Functional Progamming with Java

Einleitung:

Im Jahr 2014 wurde es mit Java 8 mit der Einführung von Streams und Lambas möglich funktional zu Programmieren.

Doch noch heute finden sich in vielen Programmen die älteren Ansätze der Imperativen Programmierung. Die oft mit einem grösseren Aufwand verbunden sind, als wenn man den Deklarativen Ansatz der Funktionalen Programmierung anwenden würde.

Mit Hilfe eines ersten Tutorials erfahren Sie die verschiedenen Aspekte der Funktionalen Programmierung unter Java.

- Java Stream
- Filter, Map, Collect, Reduce, Optional
- Anonyme Klassen
- Lambda

Ziel

- ⇨ Sie können eine Imperative Programmierung in eine funktionale Programmierung überführen.
- ⇨ Sie können Java Streams anwenden mit Filter, Map, Collect, Reduce und Optional
- ⇨ Sie können Anonyme Klassen und Lambdas anwenden.

Inhalte

Erste Gegenüberstellung

Aus einer Liste von Namen sollen alle die NICHT Peter heissen aufgelistet werden.

```
List<String> stringList = Arrays.asList("Peter", "Hans", "Rüdiger", "Werner");
```

Imperatives Programmieren:

```
/* klassische Anwendung mit einer foreach Schleife
 * und einem Filter, der den Namen Peter ausschliesst.
 */
for(String s : stringList){
   if(!"Peter".equals(s)){
      System.out.println(s);
   }
}
```

Deklarativer Ansatz mit anonymer Klasse:

```
/* Anwenden von stream und filter.
 * Implementierung des Functional Interfaces Predicate
 * mit einer anonymen Klasse für das Filtern.
 * Ausgabe auf die Konsole über eine Lamda
 */
stringList.stream().filter(new Predicate<String>() {
   @Override
   public boolean test(String s) {
      return !"Peter".equals(s);
   }
}).forEach(s -> System.out.println(s));
```

Deklarativer Ansatz mit Lambda:

```
/* Das Anwenden von Stream, filter, lamda und Funtional-Referenz
 * führt zu einer sehr kompakten Lösung
 */
stringList.stream().filter((s) ->
   !"Peter".equals(s)).forEach(System.out::println);
```

Aufgabe Erste Umsetzungen der Funktionalen Programmierung.

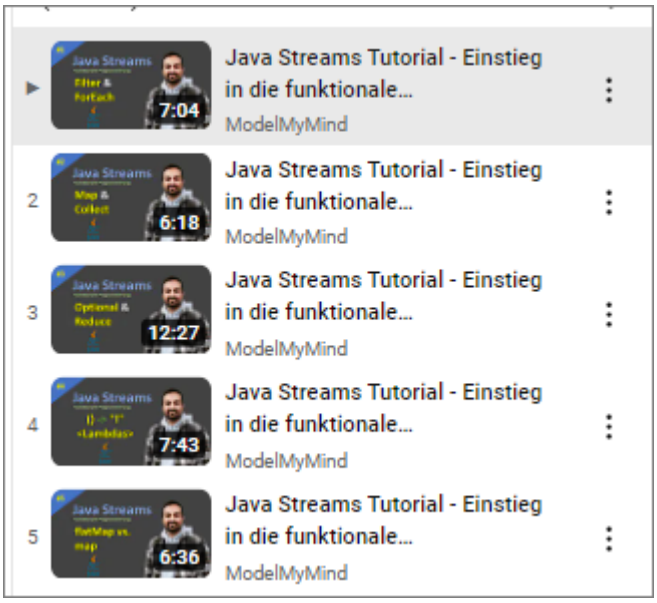
Auf dem Kanal von ModelMyMind finden Sie in fünf kurzen Tutorials eine erste Einführung in des Funktionale Programmiern mit Java.

In den Tutorials werden die verschiedensten Aspekte anhand von Beispielen erklärt.

- Java Stream
- Filter, Map, Collect, Reduce, Optional
- Anonyme Klassen
- Lambda

Aufgabe:

- Setzen Sie die fünf Tutorials anhand der Videos selber um.
  Halten Sie Ihre Implementierung in einem eigenen Projekt fest.



https://www.youtube.com/watch?v=UZpuInfZxNs&list=PLpiIfc1A7RhEaTGkA9velxuFo8S9SbyQ_&index=2

Inhalte des Tutorials:

1. Filter & ForEach
2. Map & Collect
3. Reduce & Optional
4. Lambda Ausdrücke
5. flatMap & map

(Minimum Tutorial 1-3, besser 1-5)

Ideen für Verbesserungen zum Tutorial

Statt *Arrays.asList(..* direkter mit *List.of(..*

Statt … .collect(Collectors.toList()); direkter mit *… .toList();*

Aufgabe Können Sie die Inhalte des Tutorials erklären und anwenden?

Noch haben Sie nicht alles, was das Tutorial zeigt in der Tiefe kennengelernt.
Doch sollten Sie mit Hilfe der Ergebnisse aus dem Tutorial ein paar Sachen erklären können.

1. Ausgangslage ist diese Liste:

   ```
   List<String> stringList = List.of("Apfel", "Birne", "Orange", "Quitte");
   ```

   Erklären Sie die groben Schritte, wenn Sie nur "Apfel" und "Orange" ausgeben wollen:
   - mit einem *For-Loop*
   - mit einem *Stream*

2. Schlagen Sie im Internet nach, was das *predicate interface* bei Java ist.
   - Erklären Sie in wenigen Worten das Interface.
   - Nennen Sie den *Namen der abstrakten Methode* des Interfaces
   - Nennen Sie den *Type des return-Wertes* des Interfaces

3. ACHTUNG Java kennt sowohl die *Collection Map* wie auch die *Stream-Methode map*.
   Hier geht es um die Stream Methode.
   - Erklären Sie, was die *map Methode* macht.
   - Erklären Sie was der *Type des return Wertes* der *map-Methode* ist.
   - Erklären Sie, was der Unterschied zwischen den *Methode stream* und der *Methode map* ist.

4. Ausgangslage ist diese Liste

   ```
   List<String> memberList = List.of("Muster", "Keller", "Winkler");
   ```

   Das Ergebnis auf der Console soll sein:

   ```
   Muster@lernende.bbw.ch
   Keller@lernende.bbw.ch
   Winkler@lernende.bbw.ch
   ```

   Verwenden Sie, *stream, map* und *forEach(System.out::println)* und wandeln Sie damit den Liste in den geforderten Output um.

5. Gleich wie 4, ausser dass die Ergebnisse in einer *neuen Collection* gesammelt werden müssen. (Anwenden von *collect oder direkt mit toList*)
   Erwartetes Ergebnis:
   `[Muster@lernende.bbw.ch, Keller@lernende.bbw.ch, Winkler@lernende.bbw.ch]`

6. Definiere Sie eine Liste mit den Zahlen 1, 2, 3, 4.
   Berechnen Sie mit der Liste, *stream* und mit Hilfe der Methode *reduce* die Fakultät von 4 und geben das Ergebnis auf die Konsole aus.

7. Für diese Aufgabe brauchen Sie die neuen Funktionen: *Stream.iterate* und *limit*
   int n = 4;
   Anstelle, dass Sie die Liste vorgehen, generieren Sie die Liste der Zahlen 1 bis n, berechnen die Fakultät und geben diese auf die Konsole aus.

   Es gibt noch eine andere Möglichkeit über IntStream.. finden Sie heraus, wie die geht.