

Programmieren III (Java)

4. Praktikum: Streams

Wintersemester 2023

Christopher Auer



Lernziele

- ▶ Arbeiten mit *Streams*
 - ▶ Erzeugen
 - ▶ Abbilden
 - ▶ Terminieren
- ▶ Aggregieren von Daten mit `collect`

Hinweise

- ▶ Die Praktika sind eine *ausgezeichnete Prüfungsvorbereitung*; aber nur, wenn Sie sie *eigenständig* bearbeiten oder es zumindest *versuchen*. Nachvollziehen der Lösungsvorschläge *reicht nicht* aus.
- ▶ Bearbeiten Sie die Aufgaben *vor* dem Praktikumstermin.
- ▶ Im Praktikum können Sie *Ihre Lösung* zeigen und *Fragen* stellen.
- ▶ Je nach *zeitlichem Verlauf*, wird während des Praktikumstermins der *Lösungsvorschlag besprochen*.
- ▶ Der *Lösungsvorschlag* wird online gestellt, nachdem *alle Gruppen* das Praktikum durchlaufen haben.

Aufgabe 1: Koans

Wie im letzten Praktikum, gibt es auch in diesem Praktikum wieder *kleine Programmieraufgaben* in Form von ☞ Koans. Dieses mal zum Thema ☞ Streams. Gehen Sie wie folgt vor:

- ▶ Erstellen Sie eine *Klasse Koans* in der sie die unten beschriebenen *Methoden* implementieren.
- ▶ *Alle Methoden* in Koans sind **public static**.
- ▶ Erstellen Sie eine *JUnit-Test-Klasse KoansTest* in der Sie die Methoden aus Koans *testen*.
- ▶ Lösen Sie die Aufgabe jeweils mit ☞ Streams. Sie benötigen für die Lösung *keine Schleifen* oder andere *Kontrollstrukturen*.
- ▶ Verwenden Sie *Lambda-Ausdrücke* für Referenzen auf *funktionale Interfaces*!

Koan distinctEntries ★★

Erstellen Sie eine *generische Methode*

```
long distinctEntries(Collection<T> entries)
```

Die Methode gibt die *Anzahl unterschiedlicher Einträge* in entries zurück, z.B.,

```
{1,2,3,4,2,3,3} -> 4  
{"a", "b", "a", "b", "c"} -> 3
```

Koan countEvenNumbers ★★

Erstellen Sie eine Methode

```
long countEvenNumbers(int[] numbers)
```

Die Methode gibt die *Anzahl gerader Zahlen* in dem Array zurück, z.B.

```
{1,2,3,4,2,3,3} -> 3
```

Koan intsFromStrings ★★

Erstellen Sie eine Methode

```
int[] intsFromStrings(String... strings)
```

Die Methode wandelt den *Array mit ☞ Strings* in einen *Arrays aus ints* um. Die ☞ Strings werden dabei *geparsed*, d.h. aus dem String wird ein **int** ausgelesen, z.B.

```
{"1", "2", "3", "4"} -> {1,2,3,4}
```

Koan randomSum ★★

Erstellen Sie eine Methode

```
double[] randomSum(int n)
```

Die Methode erzeugt einen **double-Array ansteigender Zufallszahlen mit n Einträgen**. Für die **Rückgabe `double[] r`** gilt

- ▶ `r[0]==0.0`
- ▶ `r[1]==r[0]+Math.random()`
- ▶ `r[2]==r[1]+Math.random()`
- ▶ ...

Ist z.B. $n=5$, so ist die Rückgabe bspw.

```
{0.0, 0.951, 1.686, 2.0128, 2.043}
```

Koan dotProduct ★★

Erstellen Sie eine Methode

```
double dotProduct(double[] v1, double[] v2)
```

Die Methode berechnet das **Skalarprodukt** von v_1 und v_2 . **Zur Erinnerung:** Sind $v = (v_1, v_2, \dots, v_n)$ und $w = (w_1, w_2, \dots, w_n)$ **zwei n -dimensionale Vektoren**, dann ist das **Skalarprodukt**

$$\langle v, w \rangle = \sum_{i=1}^n v_i \cdot w_i$$

Beispiel

```
dotProduct({1,2,3},{4,5,6})=4+10+18=32
```

Sie dürfen davon ausgehen, dass v_1 und v_2 **gleiche Länge** haben.

Koan stringsForLength ★★

Erstellen Sie eine Methode

```
Map<Integer,List<String>> stringsForLength(Collection<String> strings)
```

Die Methode erstellt aus `strings` eine \varnothing Map mit einer Zuordnung von **Stringlängen** zu **einer Liste von Strings aus strings mit dieser Länge**. **Beispiel** für `{"a","b","cc","dd","eee"}`:

```
1 -> {"a","b"}
2 -> {"cc","dd"}
3 -> {"eee"}
```

Koan collatzSeries ★★

Die folgenden Aufgaben **bauen aufeinander auf**. Wir beschäftigen uns mit der Collatz-Folge und der **unbewiesenen Collatz-Vermutung**. Sei $n \in \mathbb{N}$ eine natürliche Zahl, dann ergibt sich **Collatz-Folge** dieser Zahl wie folgt

- ▶ **Starte** mit n
- ▶ Ist die **vorherige Zahl gerade**, teile sie durch 2
- ▶ Ist die **vorherige Zahl ungerade**, multipliziere sie mit 3 und addiere 1

Beispiel für $n = 11$

11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, ...

Erreicht die Folge 1 *wiederholt* sie sich.

Erstellen Sie eine Methode

```
LongStream collatzSeries(int start)
```

- ▶ Die Methode erstellt einen *unbegrenzten* [LongStream](#)
- ▶ Der [LongStream](#) enthält die *Collatz-Folge* mit dem *Startwert start*

Koan collatzTruncated ★★ bis ★★

Wiederholt sich eine Zahl in der Collatz-Folge, so bleibt sie in einer *wiederholenden Folge, dem Orbit, „hängen“*. Ein *Beispiel für einen Orbit* ist 4, 2, 1, 4, 2, 1, In diesem Fall können wir die *Berechnung beenden*. Erstellen Sie eine Methode

```
LongStream collatzTruncated(int start)
```

Die Methode *verwendet* `collatzSeries` und *bricht die Folge* bei der *ersten Wiederholung einer Zahl* ab, z.B. 10, 5, 16, 8, 4, 2, 1, (4 → Abbruch). *Hinweis*: Der Koan `duplicateChecker` des letzten Praktikums erstellt ein *hier hilfreiches Prädikat*!

Koans collatzOrbit ★★

Die *unbewiesene Collatz-Vermutung* ist: Für jede *natürliche Zahl* endet die Collatz-Folge in dem *Orbit 4, 2, 1*. Erstellen Sie eine Methode

```
boolean collatzOrbit(int start)
```

Die Methode *verwendet* `collatzTruncated` und *prüft ob die Collatz-Folge* von `start` im *Orbit 4, 2, 1* endet. *Hinweis*: Es reicht zu prüfen ob 1 Teil der Folge ist

Koan collatzTrueForLimit ★★

Um die *Collatz-Vermutung* zu widerlegen, reicht es *ein n zu finden*, für das die Collatz-Folge nicht im *Orbit 4, 2, 1* landet. Erstellen Sie eine Methode

```
boolean collatzTrueForLimit(int limit)
```

Die Methode *verwendet* `collatzOrbit` und *prüft die Collatz-Vermutung* für alle Zahlen von 1 bis `limit`. *Parallelisieren* Sie die Berechnung!

Hier ein [interessanter Artikel](#) zur Collatz-Vermutung, aber *beachten Sie*: Schon viele haben sich an der *Collatz-Vermutung* versucht und die Beschäftigung damit hat [unerwartete Konsequenzen](#).

Koan fibonacciStream ☆☆

Die *Fibonacci-Folge* 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... ist wie folgt definiert:


- ▶ *Starte* mit 0, 1, d.h. $f_0 = 0$ und $f_1 = 1$
- ▶ Das *nächstes Folgenglied* ist die *Summe der beiden vorherigen Folgenglieder*, d.h. $f_k = f_{k-1} + f_{k-2}$ für $k \geq 2$

Erstellen Sie eine Methode

LongStream fibonacciStream()

Die Methode erstellt einen *unbegrenzten*  LongStream der die *Fibonacci-Folge* enthält.

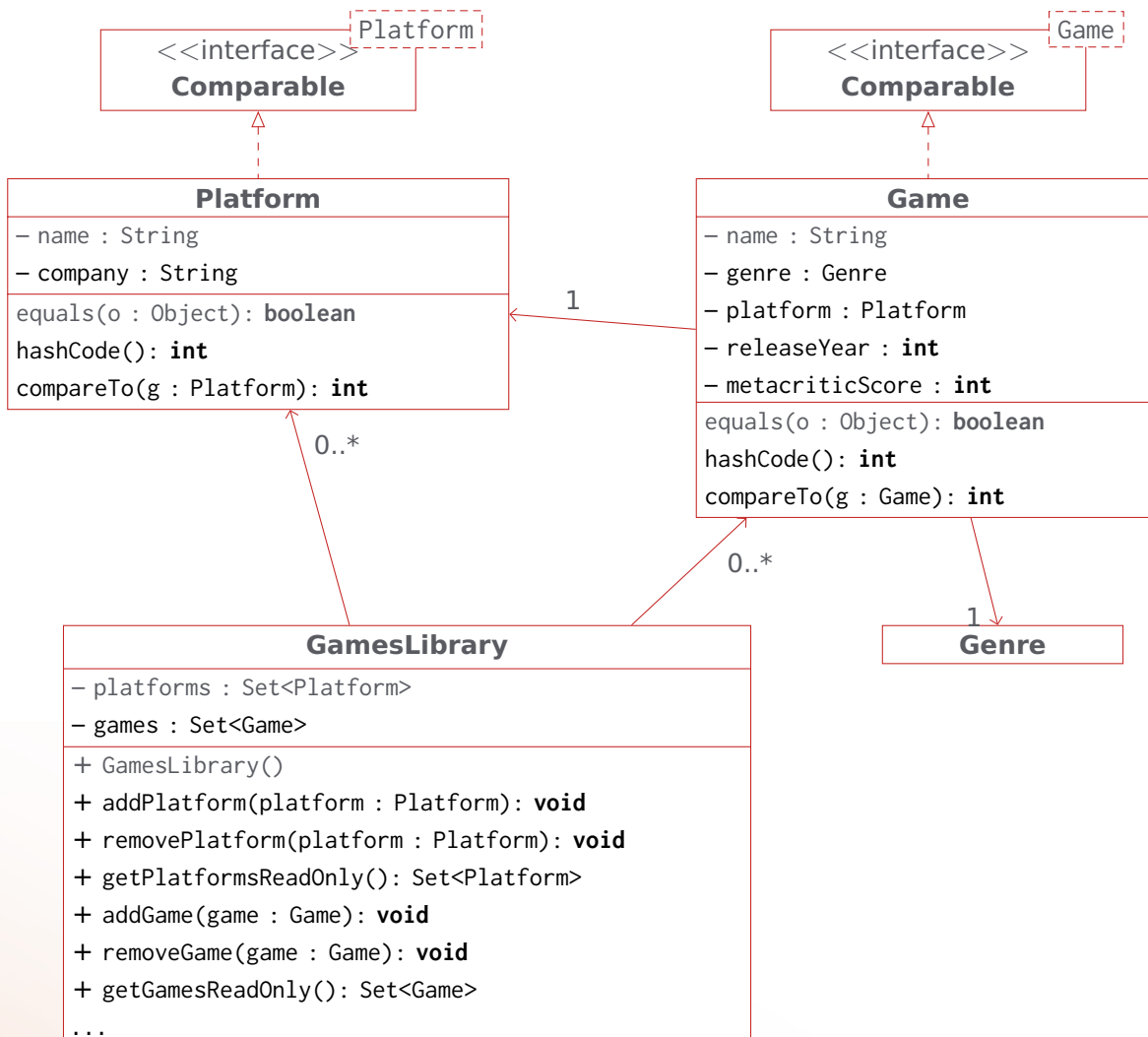
Hinweise:

- ▶ Schauen Sie die Lösung *nicht im Internet* nach!
- ▶ *Verwenden* Sie  LongStream .iterate()
- ▶ Die Funktion in iterate kann immer nur *einen Schritt* nach „*hinten schauen*“
- ▶ Für die Fibonacci-Folge müssen Sie *zwei Schritte* „*nach hinten schauen*“!



Aufgabe 2: Videospiel-Sammlung¹

Wir modellieren unsere Spielesammlung in Java. Dazu ist folgendes Klassendiagramm gegeben (aus Platzgründen ohne Konstruktoren und Getter):



- ▶ Platform modelliert eine Spieleplattform/-konsole. Die Klasse besitzt als Attribute den Hersteller company (z.B. Nintendo) und den Namen der Plattform name (z.B. Switch). Die **vollständige Implementierung** der Klasse finden Sie unter [Platform.java](#)
- ▶ Genre ist eine Enumeration mit verschiedenen Genres. Die **vollständige Implementierung** der Enumeration finden Sie unter [Genre.java](#)
- ▶ Game modelliert ein Spiel. Ein Spiel besitzt einen Namen name (z.B. „Zelda: Breath of the Wild“), ein Genre genre, modelliert durch die Enumeration Genre (z.B. Genre.ADVENTURE), eine Plattform platform (z.B. „Nintendo Switch“), ein Erscheinungsjahr releaseYear (z.B. 2017) und eine Metacritic-Score metacriticScore (z.B. 97). Die **vollständige Implementierung** der Klasse finden Sie unter [Game.java](#)
- ▶ GamesLibrary modelliert die Spielesammlung. Die Klasse verwaltet die Plattformen in Set<Platform> ↔

¹Dieses Praktikum basiert auf dem 5. Praktikum aus der Vorlesung Programmieren II aus dem Sommersemester 2020.

platforms und Spiele in `Set<Game> games`. Die Methoden zum Hinzufügen und Entfernen von Plattformen und Spielen sind bereits implementiert und unter `GamesLibrary.java` zu finden.

In `GamesLibrary.java` finden Sie eine Reihe von Methoden, die Sie in dieser Übung mit Hilfe von Streams implementieren sollen. Außerdem sollen Sie jede der Methoden testen. Dazu steht Ihnen die Testklasse `GamesLibraryTest.java` zur Verfügung. Diese beinhaltet bereits die Methode `before_createLib`, die vor jeder Ausführung einer Testmethode ausgeführt wird und eine beispielhafte Spielesammlung erstellt.

Implementieren Sie folgende Methoden in `GamesLibrary` und verwenden Sie dabei *ausschließlich Streams*:

- ▶ **Game** `getBestGame()` — gibt das Spiel mit der besten Metacritic-Score zurück.
 - ▶ **List<Game>** `sortGamesByMetacriticScore()` — gibt eine Liste aller Spiel absteigend sortiert nach der Metacritic-Score zurück.
 - ▶ **List<Game>** `sortGamesByReleaseYear()` — gibt eine Liste aller Spiele, sortiert zuerst nach dem Erscheinungsjahr und dann nach dem Namen (beides aufsteigend), zurück.
 - ▶ **double** `getAverageMetacriticScore()` — gibt die durchschnittliche Metacritic-Score aller Spiele zurück (bzw. 0 wenn sich kein Spiel in der Sammlung befindet).
 - ▶ **List<Game>** `getGamesForGenre(Genre genre)` — gibt alle Spiele aus dem Genre `genre` in einer Liste zurück.
 - ▶ **boolean** `gameReleasedBetween(int begin, int end)` — gibt **true** zurück wenn ein Spiel in den Jahren `begin` bis `end` veröffentlichte wurde (beide Schranken sind *inklusive*).
 - ▶ **Map<Platform, List<Game>>** `getGamesForPlatform()` — gibt eine `Map` zurück, die jeder Plattform ihrer Liste von Spielen zuordnet.
 - ▶ **Platform** `getBestPlatform()` — gibt die Plattform zurück, die die Spiele mit der höchsten durchschnittlichen Metacritic-Score beinhaltet.
- Hinweis:* Sie können für die Implementierung `getGamesForPlatform` nutzen.
- ▶ **List<Genre>** `getGenresWithGames()` — liefert die Genres zurück, für die mindestens ein Spiel in der Sammlung existiert.
 - ▶ **List<Game>** `topGames(long n)` — liefert die Top-n-Spiele absteigend sortiert nach Metacritic-Score zurück. Sollten weniger als `n` Spiele vorhanden sein, werden nur diese absteigend sortiert zurückgeliefert.
 - ▶ **String** `getTopTenListString()` — Erstellt einen String, der die Namen der 10 Spiele, durch Komma getrennt, beinhaltet.
 - ▶ **Map<Genre, Long>** `getGenreCount()` — Erstellt eine `Map`, die für jedes Genres die Anzahl der Spiele in dem Genre beinhaltet.
 - ▶ **Map<Platform, Double>** `getAverageScoreForPlatform()` — Erstellt eine `Map`, die für jede Plattform die mittlere Metacritic-Score aller Spiele zur Plattform beinhaltet.