SMART INDUSTRY CAMPUS

Einblick in Streams

SMART INDUSTRY CAMPUS

Was sind Streams?

- eine Datenstruktur
- zum effizienten Verarbeiten von Daten
- ein Stream ist eine Folge von Daten, die aus einer Quelle (z.B.: List, Map, Set, ...) stammen
- auf diesen Stream kann man verschiedene Operationen durchführen:
  - Filtern
  - Sortieren
  - Sammeln

Alltagsbeispiel

- Beispiel:
  - Wir haben eine Liste an Personen und wollen sie nach bestimmten Kriterien sortieren
- Mittels Stream-API simpel lösbar!

Wie erstellt man einen Stream?

- .stream():
  - Eine Datenquelle wird in einen Stream „verwandelt“

Verwandeln einer List in einen Stream

Es gibt IntStream, DoubleStream für Arrays

```
import java.util.*;
import java.util.stream.*;

public class StreamErstellen {
    public static void main(String[] args) {
        List<String> namen = Arrays.asList("Anna", "Bernd", "Clara", "David");
        Stream<String> nameStream = namen.stream();

        int[] zahlen = {1, 2, 3, 4, 5};
        IntStream zahlenStream = Arrays.stream(zahlen);

    }
}
```

Filtern eines Streams

- .filter(Predicate predicate):
  - Ein Stream wird nach Bedingung gefiltet

Verwandeln einer List in einen Stream

```
List<String> namen = Arrays.asList("Anna", "Bernd", "Clara", "David");
namen.stream()
        .filter(name -> name.startsWith("A"))
        .forEach(System.out::println); // Ausgabe: Anna
```

Alle Namen, für die die Bedingung falsch ist, fliegen raus.

Aufgabe

Erstelle eine Liste von Wörtern und filtere alle Wörter, die mit "B" beginnen. Nutze dafür Streams.

SMART INDUSTRY CAMPUS

Aufgabe

Erstelle eine Liste, welche die Nummern 1- 10 enthält. Erstelle eine erneute Liste mit allen geraden Zahlen. Nutze dafür Streams.

SMART INDUSTRY CAMPUS

Alltagsbeispiel

- Beispiel:
  - Wir sind in einer Bibliothek
  - Wir wollen alle Bücher finden, die von einem bestimmten Autor sind.
  - Die Bücher wollen wir nach Erscheinungsjahr ausgegeben bekommen
  - Es soll nur der Titel ausgegeben werden

- Mittels Stream-API simpel lösbar!

SMART INDUSTRY CAMPUS

Beispiel ohne Stream

Beisp

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Buch> buecher = new ArrayList<Buch>();
        buecher.add(new Buch("Buch A", "Autor X", 2001));
        buecher.add(new Buch("Buch B", "Autor Y", 1995));
        buecher.add(new Buch("Buch C", "Autor X", 2010));

        List<Buch> gefilterteBuecher = new ArrayList<Buch>();
        for (Buch buch : buecher) {
            if (buch.autor.equals("Autor X")) {
                gefilterteBuecher.add(buch);
            }
        }

        Collections.sort(gefilterteBuecher, new Comparator<Buch>() {
            @Override
            public int compare(Buch b1, Buch b2) {
                return Integer.compare(b1.erscheinungsjahr, b2.erscheinungsjahr);
            }
        });

        List<String> titelListe = new ArrayList<String>();
        for (Buch buch : gefilterteBuecher) {
            titelListe.add(buch.titel);
        }

        System.out.println(titelListe); // Ausgabe: [Buch A, Buch C]
    }
}
```

```java
class Buch {
    String titel;
    String autor;
    int erscheinungsjahr;

    Buch(String titel, String autor, int erscheinungsjahr) {
        this.titel = titel;
        this.autor = autor;
        this.erscheinungsjahr = erscheinungsjahr;
    }
}
```

SMART INDUSTRY CAMPUS

Beispiel mit Lambdas

SMART INDUSTRY CAMPUS

Beispiel mit Lambdas

```java
import java.util.ArrayList;
import java.util.List;

class MainLambda {
    public static void main(String[] args) {
        List<Buch> buecher = new ArrayList<Buch>();
        buecher.add(new Buch("Buch A", "Autor X", 2001));
        buecher.add(new Buch("Buch B", "Autor Y", 1995));
        buecher.add(new Buch("Buch C", "Autor X", 2010));

        List<Buch> gefilterteBuecher = new ArrayList<Buch>();
        for (Buch buch : buecher) {
            if (buch.autor.equals("Autor X")) {
                gefilterteBuecher.add(buch);
            }
        }

        gefilterteBuecher.sort((b1, b2) -> Integer.compare(b1.erscheinungsjahr, b2.erscheinungsjahr));

        List<String> titelListe = new ArrayList<String>();
        gefilterteBuecher.forEach(buch -> titelListe.add(buch.titel));

        System.out.println(titelListe); // Ausgabe: [Buch A, Buch C]
    }
}
```

```java
class Buch {
    String titel;
    String autor;
    int erscheinungsjahr;

    Buch(String titel, String autor, int erscheinungsjahr) {
        this.titel = titel;
        this.autor = autor;
        this.erscheinungsjahr = erscheinungsjahr;
    }
}
```

SMART INDUSTRY CAMPUS

Beispiel mit Stream

Beispiel mit Lambdas

```java
class Buch {
    String titel;
    String autor;
    int erscheinungsjahr;

    Buch(String titel, String autor, int erscheinungsjahr) {
        this.titel = titel;
        this.autor = autor;
        this.erscheinungsjahr = erscheinungsjahr;
    }
```

```java
public static void main(String[] args) {
    List<Buch> buecher = Arrays.asList(
            new Buch("Buch A", "Autor X", 2001),
            new Buch("Buch B", "Autor Y", 1995),
            new Buch("Buch C", "Autor X", 2010)
    );

    List<String> result = buecher.stream()
            .filter(b -> b.autor.equals("Autor X")) // Filtern
            .sorted(Comparator.comparingInt(b -> b.erscheinungsjahr)) // Sortieren
            .map(b -> b.titel) // Sammeln
            .collect(Collectors.toList());

    System.out.println(result); // Ausgabe: [Buch A, Buch C]
}
```

Was macht der Code?

- .stream():
  - Aus unserer Quelle wird ein Stream gebildet
  - Dieser Stream ermöglicht uns, die Bücher zu „durchschauen“

- .filter(Predicate predicate):
  - Gibt einen Stream zurück, welches alle Bücher enthält, die den Autor X haben

```
public static void main(String[] args) {
    List<Buch> buecher = Arrays.asList(
            new Buch("Buch A", "Autor X", 2001),
            new Buch("Buch B", "Autor Y", 1995),
            new Buch("Buch C", "Autor X", 2010)
    );

    List<String> result = buecher.stream()
            .filter(b -> b.autor.equals("Autor X")) // Filtern
            .sorted(Comparator.comparingInt(b -> b.erscheinungsjahr)) // Sortieren
            .map(b -> b.titel) // Sammeln
            .collect(Collectors.toList());

    System.out.println(result); // Ausgabe: [Buch A, Buch C]
}
```

Was macht der Code?

- .sorted(Comparator comparator):
  - Damit sortiert man alle Elemente nach bestimmten Kriterien
  - In unserem Fall wird der Stream nach Erscheinungsjahr sortiert
  - Comparator ist ebenfalls ein funktionales Interface
    - comparingInt ist eine statische Methode, die wiederum ein funktionales Interface als Parameter entgegennimmt
    - Es werden die Erscheinungsjahre extrahiert und der Comparator vergleicht diese

```java
public static void main(String[] args) {
    List<Buch> buecher = Arrays.asList(
            new Buch("Buch A", "Autor X", 2001),
            new Buch("Buch B", "Autor Y", 1995),
            new Buch("Buch C", "Autor X", 2010)
    );

    List<String> result = buecher.stream()
            .filter(b -> b.autor.equals("Autor X")) // Filtern
            .sorted(Comparator.comparingInt(b -> b.erscheinungsjahr)) // Sortieren
            .map(b -> b.titel) // Sammeln
            .collect(Collectors.toList());

    System.out.println(result); // Ausgabe: [Buch A, Buch C]
}
```

.sorted() sortiert bspw. alphabetisch oder Zahlen nach ihrer Reihenfolge

Was macht der Code?

- .map(Function mapper):
  - Verwandelt jedes Element in ein anderes Element anhand übergebener Funktion
  - In unserem Beispiel wird jedes Buch-Objekt in seinen Titel umgewandelt
    - Man erhält einen Stream aus Strings statt Buch

```
public static void main(String[] args) {
    List<Buch> buecher = Arrays.asList(
            new Buch("Buch A", "Autor X", 2001),
            new Buch("Buch B", "Autor Y", 1995),
            new Buch("Buch C", "Autor X", 2010)
    );

    List<String> result = buecher.stream()
            .filter(b -> b.autor.equals("Autor X")) // Filtern
            .sorted(Comparator.comparingInt(b -> b.erscheinungsjahr)) // Sortieren
            .map(b -> b.titel) // Sammeln
            .collect(Collectors.toList());

    System.out.println(result); // Ausgabe: [Buch A, Buch C]
}
```

Was macht der Code?

- .collect(Collector collector):
  - Der Stream wird wieder in eine Collection (Liste,...) umgewandelt

```
public static void main(String[] args) {
    List<Buch> buecher = Arrays.asList(
            new Buch("Buch A", "Autor X", 2001),
            new Buch("Buch B", "Autor Y", 1995),
            new Buch("Buch C", "Autor X", 2010)
    );

    List<String> result = buecher.stream()
            .filter(b -> b.autor.equals("Autor X")) // Filtern
            .sorted(Comparator.comparingInt(b -> b.erscheinungsjahr)) // Sortieren
            .map(b -> b.titel) // Sammeln
            .collect(Collectors.toList());

    System.out.println(result); // Ausgabe: [Buch A, Buch C]
}
```

Aufgabe

Sortiere eine Liste von Namen alphabetisch und gib sie aus.

Nutze die Methode:

- sorted()

Aufgabe

Wandelt eine Liste von Temperaturen in Celsius in Fahrenheit um (map).

Aufgabe

Berechne die Summe aller geraden Zahlen in einer Liste.

Nutze dafür folgende Methoden:

- .mapToInt(Functional functional)
- .sum()

SMART INDUSTRY CAMPUS

Aufgabe

Du hast eine Liste von Filmen. Jeder Film hat einen Titel, ein Erscheinungsjahr und einen Regisseur. Finde alle Filme, die nach dem Jahr 2000 veröffentlicht wurden, sortiere diese Filme nach ihrem Erscheinungsjahr und gib den Titel der Filme aus.

```
List<Film> filme = new ArrayList<Film>();
        filme.add(new Film("The Shawshank Redemption", 1994));
        filme.add(new Film("The Dark Knight", 2008));
        filme.add(new Film("Pulp Fiction", 1994));
        filme.add(new Film("The Lord of the Rings: The Return of the King", 2003));
        filme.add(new Film("Inception", 2010));
        filme.add(new Film("Fight Club", 1999));
        filme.add(new Film("Forrest Gump", 1994));
        filme.add(new Film("The Matrix", 1999));
        filme.add(new Film("Gladiator", 2000));
        filme.add(new Film("The Avengers", 2012));
```