# *Kapitel 9: Streams und Lambda Expressions*

*Lernziele*

[LZ 9.1] Erklären können, was ein Lambda Ausdruck ist
[LZ 9.2] Funktionale Interfaces definieren und anwenden können
[LZ 9.3] Wichtige Funktionale Interfaces kennen
[LZ 9.4] Lambdas zur Filterung und Verarbeitung von Daten einsetzen können
[LZ 9.5] Erklären können, was ein Stream ist und wie er arbeitet
[LZ 9.6] Wichtige create-, intermediate- und terminal-Operationen kennen und diese anwenden können

# 9. Streams und Lambda Expressions

*[LZ 9.1] Erklären können, was ein Lambda Ausdruck ist*

- Erläutern Sie den Begriff des Lambda Ausdrucks anhand einiger Beispiele

# 9. Streams und Lambda Expressions

*[LZ 9.2] Funktionale Interfaces definieren und anwenden können*

- Erklären Sie den Begriff „Funktionales Interface“ und geben Sie ein Beispiel in Java an.
- Definieren Sie ein funktionales Interface und rufen Sie dieses auf (analog zu MathOperation im Skript!)
- Gegeben Sei folgendes Interface:

```
public interface Function {
        long eval(final long left, final long right);
}
```

Nachfolgend sind einige Lambdas gezeigt. Welche davon sind gültig? Wofür erhält man Fehler beim Kompilieren?

```
final Function v1 = (long x, Long y) -> { return x + y; };
final Function v2 = (long x, long y) -> { return x + y; };
final Function v3 = (long x, long y) -> x + y;
final Function v4 = (long x, y) -> x + y;
final Function v5 = (x, y) -> x + y;
final Function v6 = x, y -> x + y;
```

- Schreibe ein Functional Interface „StringToIntConverter“ für die Abbildung von String-Objekten auf int und implementiere diese mit mindestens einem Lambda.

- Geben Sie die Definitionen der Ihnen bekannten funktionalen Interfaces an!

- Gegeben sei die Personenbearbeitung aus dem Skript:

```
Predicate<Person> checkAlter = p -> p.getAlter() >= 18;
Consumer<Person> printPerson = p -> System.out.println(p);
processPersons(personen, checkAlter, printPerson);
```

- Geben Sie eigene Lambdas an, mit denen das Verhalten entsprechend geändert werden kann!
- Geben Sie eine Methode

```
List<Person> filterPersons(List<Person> liste, Predicate<Person> tester)
```

an, die eine Liste zurückliefert, in der nur die Personen enthalten sind, die das Prädikat erfüllen!

# *9. Streams und Lambda Expressions*

*[LZ 9.5] Erklären können, was ein Stream ist und wie er arbeitet*

- Erläutern Sie den Begriff und die Arbeitsweise eines „Streams“ anhand eines Beispiels

# 9. Streams und Lambda Expressions

*[LZ 9.6] Wichtige create-, intermediate- und terminal-Operationen kennen und diese anwenden können*

- Welche create-Operationen kennen Sie?
- Welche intermediate-Operationen kennen Sie?
- Welche terminal-Operationen kennen Sie?
- Zeigen Sie den Einsatz obiger Operationen (mindestens eine von jedem Typ) anhand eines Beispiels!

Gegeben sei:

```
Stream.of("Andi", "Barbara", "Carsten", "Marius", "Micha", "Tim")
.map(name -> {
        System.out.println("map: " + name);
        return name.toUpperCase();
})
.filter(name -> {
        System.out.println("filter: " + name);
        return name.startsWith("M");
})
.forEach(name -> System.out.println("forEach: " + name));
```

- Welche Ausgaben produziert obiger Code?
- In welcher Reihenfolge sollten die Operationen ausgeführt werden?
- Tausche dazu map() und filter()!