Funktionales Programmieren mit Java

Andreas Jürgensen

Vortrag::learnings

Durch diesen Vortrag werdet ihr lernen…

… was Funktionale Programmierung ist.

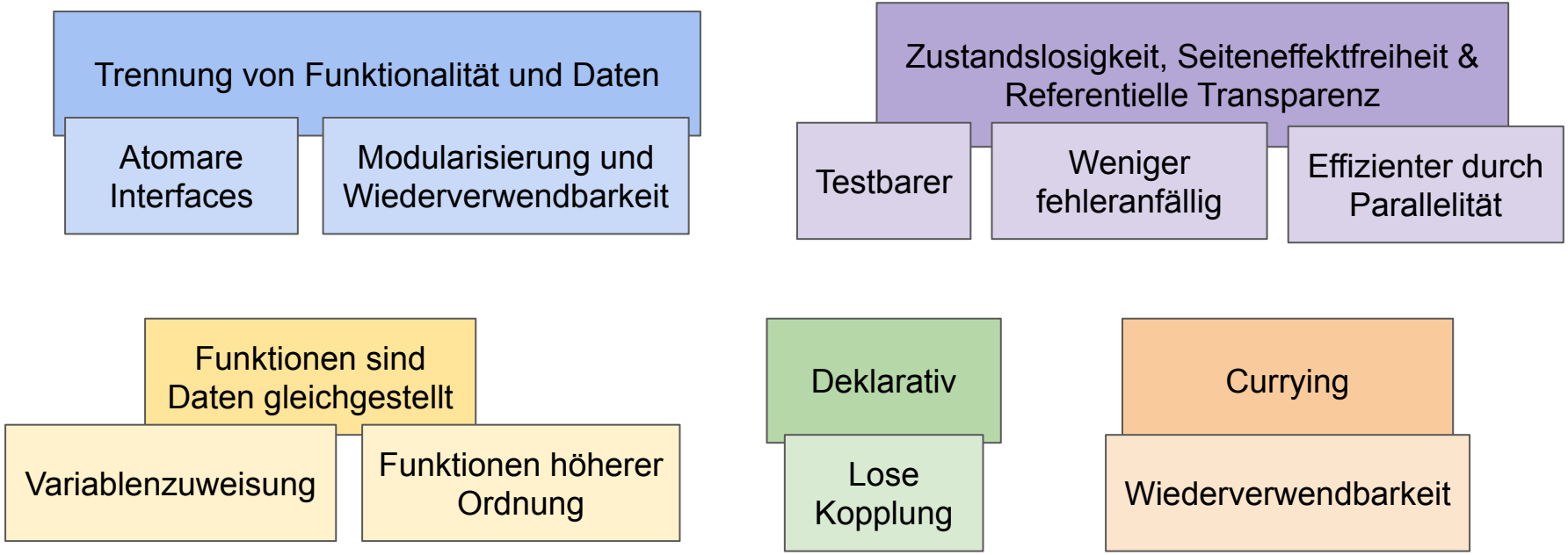
… wie mit Java funktional programmiert werden kann.

… wie Funktionale Programmierung in Java zur Performance-Optimierung eingesetzt werden kann.

… wann welches Programmierparadigma verwendet werden sollte.

… wie funktional programmierte Komponenten in objektorientierten Programme eingebunden werden sollten.

funktionaleProgrammierung.prinzipien().map(Prinzip::vorteile)



```java
import java.util.function.BinaryOperator;


interface Multiplication extends BinaryOperator<Double> {
   Multiplication PRODUCT = (multiplier, multiplicand) -> multiplier * multiplicand;
}
```

```java
import java.util.function.BinaryOperator;


interface Multiplication extends BinaryOperator<Double> {


}
```

```java
import java.util.function.BinaryOperator;


interface Multiplication extends BinaryOperator<Double> {


}
```

```java
/**
* Represents an operation upon two operands of the same type, producing a result
* of the same type as the operands.  This is a specialization of
* BiFunction for the case where the operands and the result are all of
* the same type.
*
* This is a functional interface whose functional method is #apply(Object, Object).
*
* @param <T> the type of the operands and result of the operator
*
* @see BiFunction
* @see UnaryOperator
* @since 1.8
*/
```

```java
import java.util.function.BinaryOperator;


interface Multiplication extends BinaryOperator<Double> {
   Multiplication PRODUCT = (multiplier, multiplicand) -> multiplier * multiplicand;
}


double answer = PRODUCT.apply(2.0, 21.0);
```

```java
import java.util.function.Function;
import java.util.function.UnaryOperator;

interface Multiplication extends Function<Double, UnaryOperator<Double>> {

}
```

```java
import java.util.function.Function;
import java.util.function.UnaryOperator;

interface Multiplication extends Function<Double, UnaryOperator<Double>> {
   Multiplication PRODUCT = multiplier -> multiplicand -> multiplier * multiplicand;
}
```

```java
import java.util.function.Function;
import java.util.function.UnaryOperator;

interface Multiplication extends Function<Double, UnaryOperator<Double>> {
   Multiplication PRODUCT = multiplier -> multiplicand -> multiplier * multiplicand;
}

double answer = PRODUCT.apply(2.0).apply(21.0);
```

```java
import java.util.function.Function;
import java.util.function.UnaryOperator;

interface Multiplication extends Function<Double, UnaryOperator<Double>> {
   Multiplication PRODUCT = multiplier -> multiplicand -> multiplier * multiplicand;
}

double answer = PRODUCT.apply(2.0).apply(21.0);

interface Doubling extends UnaryOperator<Double> {

}
```

```java
import java.util.function.Function;
import java.util.function.UnaryOperator;

interface Multiplication extends Function<Double, UnaryOperator<Double>> {
   Multiplication PRODUCT = multiplier -> multiplicand -> multiplier * multiplicand;
}

double answer = PRODUCT.apply(2.0).apply(21.0);

interface Doubling extends UnaryOperator<Double> {
   Doubling DOUBLE = PRODUCT.apply(2.0)::apply;
}
```

```java
import java.util.function.Function;
import java.util.function.UnaryOperator;

interface Multiplication extends Function<Double, UnaryOperator<Double>> {
   Multiplication PRODUCT = multiplier -> multiplicand -> multiplier * multiplicand;
}

double answer = PRODUCT.apply(2.0).apply(21.0);

interface Doubling extends UnaryOperator<Double> {
   Doubling DOUBLE = PRODUCT.apply(2.0)::apply;
}

double sameAnswer = DOUBLE.apply(21.0);
```

```java
import java.util.function.Function;
import java.util.function.UnaryOperator;

interface Multiplication extends Function<Double, UnaryOperator<Double>> {
   Multiplication PRODUCT = multiplier -> multiplicand -> multiplier * multiplicand;
}

double answer = PRODUCT.apply(2.0).apply(21.0);

interface Doubling extends UnaryOperator<Double> {
   Doubling DOUBLE = PRODUCT.apply(2.0)::apply;
}

double sameAnswer = DOUBLE.apply(21.0);
```

Vortrag::learnings

Durch diesen Vortrag werdet ihr lernen…

… was Funktionale Programmierung ist.

… wie mit Java funktional programmiert werden kann.

… wie Funktionale Programmierung in Java zur Performance-Optimierung eingesetzt werden kann.

… wann welches Programmierparadigma verwendet werden sollte.

… wie funktional programmierte Komponenten in objektorientierten Programme eingebunden werden sollten.

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {

}
```

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {

    }
}
```

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;



    }
}
```

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;





    }
}
```

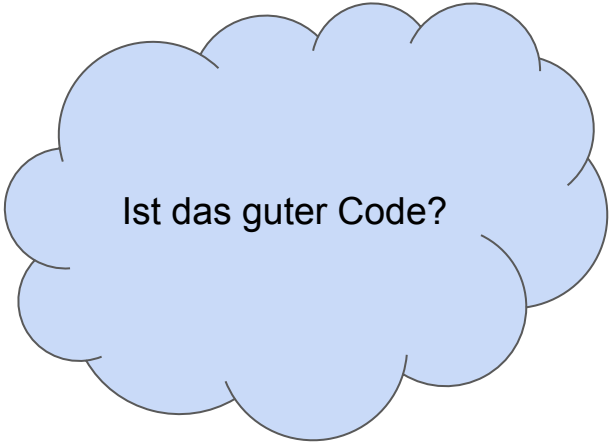
```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;



    }
}
```

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```



```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

Single-Responsibility?

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

Single-Responsibility?

Welche Formen gibt es?

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

Single-Responsibility?

Welche Formen gibt es?

Für welche Form soll die Fläche berechnet werden?

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

Single-Responsibility?

Welche Formen gibt es?

Für welche Form soll die Fläche berechnet werden?

Interpretation des Parameters `sizeConfiguration`

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

Single-Responsibility?

Welche Formen gibt es?

Für welche Form soll die Fläche berechnet werden?

Interpretation des Parameters `sizeConfiguration`

Wie wird die Fläche eines Rechtecks, Quadrats und Kreises berechnet?

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

Single-Responsibility?

Welche Formen gibt es?

Für welche Form soll die Fläche berechnet werden?

Interpretation des Parameters `sizeConfiguration`

Wie wird die Fläche eines Rechtecks, Quadrats und Kreises berechnet?

Umgang mit verschiedenen Formen

```java
double calculateArea(final String shapeName, final Double[] sizeConfiguration) {
    switch (shapeName) {
        case "rectangle":
            double width = sizeConfiguration[0];
            double height = sizeConfiguration[1];
            return width * height;
        case "square":
            double length = sizeConfiguration[0];
            return length * length;
        case "circle":
            double radius = sizeConfiguration[0];
            return Math.PI * radius * radius;
        default:
            throw new IllegalArgumentException("no such shape");
    }
}
```

Single-Responsibility?

Welche Formen gibt es?

Für welche Form soll die Fläche berechnet werden?

Interpretation des Parameters `sizeConfiguration`

Wie wird die Fläche eines Rechtecks, Quadrats und Kreises berechnet?

Umgang mit verschiedenen Formen

Fehlerbehandlung

```java
sealed interface Shape permits Rectangle, Square, Circle { }
```

Welche Formen gibt es?

```java
sealed interface Shape permits Rectangle, Square, Circle { }

record Rectangle(double length, double width) implements Shape { }

record Square(double length) implements Shape { }

record Circle(double radius) implements Shape { }
```

Welche Formen gibt es?

```
interface RectangleCreation extends Function<Double[], Rectangle> {



}
```

Interpretation des Parameters **`sizeConfiguration`**

```java
interface RectangleCreation extends Function<Double[], Rectangle> {
   RectangleCreation RECTANGLE = sizeConfiguration ->
           new Rectangle(sizeConfiguration[0], sizeConfiguration[1]);
}
```

Interpretation des Parameters **`sizeConfiguration`**

```java
interface RectangleCreation extends Function<Double[], Rectangle> {
   RectangleCreation RECTANGLE = sizeConfiguration ->
           new Rectangle(sizeConfiguration[0], sizeConfiguration[1]);
}

interface SquareCreation extends Function<Double[], Square> {
   SquareCreation SQUARE = sizeConfiguration ->
           new Square(sizeConfiguration[0]);
}

interface CircleCreation extends Function<Double[], Circle> {
   CircleCreation CIRCLE = sizeConfiguration ->
           new Circle(sizeConfiguration[0]);
}
```

Interpretation des Parameters `sizeConfiguration`

```
interface ShapeCreation extends Function<String, Function<Double[], Shape>> {

}
```

Umgang mit verschiedenen Formen

Fehlerbehandlung

```java
interface ShapeCreation extends Function<String, Function<Double[], Shape>> {
    ShapeCreation SHAPE = shapeName -> switch (shapeName) {
        case "rectangle" -> RECTANGLE::apply;
        case "square" -> SQUARE::apply;
        case "circle" -> CIRCLE::apply;
        default -> throw new IllegalArgumentException("no such shape");
    };
}
```

Umgang mit verschiedenen Formen

Fehlerbehandlung

```java
interface AreaOfRectangleCalculation extends Function<Rectangle, Double> {



}
```

Wie wird die Fläche eines Rechtecks, Quadrats und Kreises berechnet?

```java
interface AreaOfRectangleCalculation extends Function<Rectangle, Double> {
   AreaOfRectangleCalculation AREA_OF_RECTANGLE =
           rectangle -> rectangle.length() * rectangle.width();
}
```

Wie wird die Fläche eines Rechtecks, Quadrats und Kreises berechnet?

```java
interface AreaOfRectangleCalculation extends Function<Rectangle, Double> {
   AreaOfRectangleCalculation AREA_OF_RECTANGLE =
           rectangle -> rectangle.length() * rectangle.width();
}

interface AreaOfSquareCalculation extends Function<Square, Double> {
   AreaOfSquareCalculation AREA_OF_SQUARE =
           square -> square.length() * square.length();
}

interface AreaOfCircleCalculation extends Function<Circle, Double> {
   AreaOfCircleCalculation AREA_OF_CIRCLE =
           circle -> Math.PI * circle.radius() * circle.radius();
}
```

Wie wird die Fläche eines Rechtecks, Quadrats und Kreises berechnet?

```
interface AreaOfShapeCalculation extends Function<Shape, Double> {



}
```

Für welche Form soll die Fläche berechnet werden?

```java
interface AreaOfShapeCalculation extends Function<Shape, Double> {
   AreaOfShapeCalculation AREA = shape -> switch (shape) {
       case Rectangle rectangle -> AREA_OF_RECTANGLE.apply(rectangle);
       case Square square -> AREA_OF_SQUARE.apply(square);
       case Circle circle -> AREA_OF_CIRCLE.apply(circle);
   };
}
```

Für welche Form soll die Fläche berechnet werden?

```java
class AreaCalculationProcedural {
   double calculateArea(String shapeName, Double[] sizeConfiguration) {
      // …
   }
}
```

```java
class AreaCalculationProcedural {
   double calculateArea(String shapeName, Double[] sizeConfiguration) {
      // …
   }
}


interface AreaCalculationFunctional extends BiFunction<String, Double[], Double> {
   AreaCalculationFunctional CALCULATED_AREA = (shapeName, sizeConfiguration) ->

}
```

```java
class AreaCalculationProcedural {

   double calculateArea(String shapeName, Double[] sizeConfiguration) {

      // …

   }

}


interface AreaCalculationFunctional extends BiFunction<String, Double[], Double> {
   AreaCalculationFunctional CALCULATED_AREA = (shapeName, sizeConfiguration) ->
           AREA.apply(                                                        );
}
```

```java
class AreaCalculationProcedural {

   double calculateArea(String shapeName, Double[] sizeConfiguration) {

      // …

   }

}


interface AreaCalculationFunctional extends BiFunction<String, Double[], Double> {
   AreaCalculationFunctional CALCULATED_AREA = (shapeName, sizeConfiguration) ->
           AREA.apply(SHAPE                                                  );
}
```

```java
class AreaCalculationProcedural {

   double calculateArea(String shapeName, Double[] sizeConfiguration) {

      // …

   }

}


interface AreaCalculationFunctional extends BiFunction<String, Double[], Double> {
   AreaCalculationFunctional CALCULATED_AREA = (shapeName, sizeConfiguration) ->
           AREA.apply(SHAPE.apply(shapeName)                                  );
}
```

```java
class AreaCalculationProcedural {

   double calculateArea(String shapeName, Double[] sizeConfiguration) {

      // …

   }

}


interface AreaCalculationFunctional extends BiFunction<String, Double[], Double> {
   AreaCalculationFunctional CALCULATED_AREA = (shapeName, sizeConfiguration) ->
           AREA.apply(SHAPE.apply(shapeName).apply(sizeConfiguration));
}
```

Vortrag::learnings

Durch diesen Vortrag werdet ihr lernen…

… was Funktionale Programmierung ist.

… wie mit Java funktional programmiert werden kann.

… wie Funktionale Programmierung in Java zur Performance-Optimierung eingesetzt werden kann.

… wann welches Programmierparadigma verwendet werden sollte.

… wie funktional programmierte Komponenten in objektorientierten Programme eingebunden werden sollten.

```java
final class CustomerRepository {
   private final Collection<Customer> allCustomers;
   Collection<Customer> customersWithMostOrders() {
      Collection<Customer> customersWithMostOrders = new ArrayList<>();
      int mostOrders = 0;
      for (Customer customer : this.allCustomers) {
         int numberOfOrdersByThisCustomer = customer.numberOfOrders();
         if (numberOfOrdersByThisCustomer > mostOrders) {
            customersWithMostOrders = new ArrayList<>(List.of(customer));
            mostOrders = numberOfOrdersByThisCustomer;
         } else if (numberOfOrdersByThisCustomer == mostOrders) {
            customersWithMostOrders.add(customer);
         }
      }
      return customersWithMostOrders;
   }
}
```

Langsam!

Objektorientiert / Prozedural

✓ SpeedTestProcedural	2 sec 51 ms
✓ fiveUsersWithTenOrdersInTotal_aSingleCustomerWithMostOrder	2 sec 51 ms

```java
interface CustomersWithMostOrders extend UnaryOperator<Collection<Customer>> {

}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {

   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->





}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {

   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->

         allCustomers.stream()









}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {

   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->

         allCustomers.stream()

                  .map(customer ->

                      new Tuple(List.of(customer), customer.numberOfOrders()))




}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {

   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->

         allCustomers.stream()

                  .map(customer ->

                      new Tuple(List.of(customer), customer.numberOfOrders()))

                  .reduce(TO_CUSTOMERS_WITH_MOST_ORDERS)



}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {
   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->
         allCustomers.stream()
                  .map(customer ->
                      new Tuple(List.of(customer), customer.numberOfOrders()))
                  .reduce(TO_CUSTOMERS_WITH_MOST_ORDERS)
                  .map(Tuple::customers)


}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {
   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->
         allCustomers.stream()
                  .map(customer ->
                      new Tuple(List.of(customer), customer.numberOfOrders()))
                  .reduce(TO_CUSTOMERS_WITH_MOST_ORDERS)
                  .map(Tuple::customers)
                  .orElse(List.of());
}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {
   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->
         allCustomers.stream()
                  .map(customer ->
                      new Tuple(List.of(customer), customer.numberOfOrders()))
                  .reduce(TO_CUSTOMERS_WITH_MOST_ORDERS)
                  .map(Tuple::customers)
                  .orElse(List.of());
}
```

Langsam!

Objektorientiert / Prozedural

✓ SpeedTestProcedural	2 sec 51 ms
✓ fiveUsersWithTenOrdersInTotal_aSingleCustomerWithMostOrder	2 sec 51 ms

Funktional

✓ SpeedTestFunctional	2 sec 62 ms
✓ fiveUsersWithTenOrdersInTotal_aSingleCustomerWithMostOrder	2 sec 62 ms

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {
   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->
         allCustomers.stream()
                  .map(customer ->
                      new Tuple(List.of(customer), customer.numberOfOrders()))
                  .reduce(TO_CUSTOMERS_WITH_MOST_ORDERS)
                  .map(Tuple::customers)
                  .orElse(List.of());
}
```

```java
interface CustomersWithMostOrders extends UnaryOperator<Collection<Customer>> {
   CustomersWithMostOrders CUSTOMERS_WITH_MOST_ORDERS = allCustomers ->
         allCustomers.stream()
                  .parallel()
                  .map(customer ->
                      new Tuple(List.of(customer), customer.numberOfOrders()))
                  .reduce(TO_CUSTOMERS_WITH_MOST_ORDERS)
                  .map(Tuple::customers)
                  .orElse(List.of());
}
```

Objektorientiert / Prozedural

✓ SpeedTestProcedural	2 sec 51 ms
✓ fiveUsersWithTenOrdersInTotal_aSingleCustomerWithMostOrder	2 sec 51 ms

Funktional

✓ SpeedTestFunctional	2 sec 62 ms
✓ fiveUsersWithTenOrdersInTotal_aSingleCustomerWithMostOrder	2 sec 62 ms

Funktional mit Parallelisierung

✓ SpeedTestFunctional	445 ms
✓ fiveUsersWithTenOrdersInTotal_aSingleCustomerWithMostOrders	445 ms

Vortrag::learnings

Durch diesen Vortrag werdet ihr lernen…

… was Funktionale Programmierung ist.

… wie mit Java funktional programmiert werden kann.

… wie Funktionale Programmierung in Java zur Performance-Optimierung eingesetzt werden kann.

… wann welches Programmierparadigma verwendet werden sollte.

… wie funktional programmierte Komponenten in objektorientierten Programme eingebunden werden sollten.

funktionaleProgrammierung.compareTo(oop)

Komplexe Software-Systeme

GUI-Anwendungen

Enterprise-Software und Geschäftslogik

Datenbankanwendungen

Mathematische Berechnungen

Algorithmen

Verteilte Systeme

Echtzeitsysteme

Parallelität und Nebenläufigkeit

OOP ⟷ **FP**

funktionaleProgrammierung.compareTo(oop)

Komplexe Software-Systeme	?	Mathematische Berechnungen
GUI-Anwendungen	?	Algorithmen
		Verteilte Systeme
Enterprise-Software und Geschäftslogik	?	Echtzeitsysteme
Datenbankanwendungen	?	Parallelität und Nebenläufigkeit

OOP ⟵ ⟶ **FP**

```java
Map<Integer, String> representations = new HashMap<>(ofEntries(
        entry(1, "one"),
        entry(2, "two")));
```

```java
Map<Integer, String> representations = new HashMap<>(ofEntries(
        entry(1, "one"),
        entry(2, "two")));

Consumer<List<Integer>> printIntegers = ints -> ints.stream()
        .map(representations::get)
        .forEach(System.out::println);
```

```java
Map<Integer, String> representations = new HashMap<>(ofEntries(
        entry(1, "one"),
        entry(2, "two")));

Consumer<List<Integer>> printIntegers = ints -> ints.stream()
        .map(representations::get)
        .forEach(System.out::println);

printIntegers.accept(List.of(1, 2, 3));
```

```java
Map<Integer, String> representations = new HashMap<>(ofEntries(
        entry(1, "one"),
        entry(2, "two")));

Consumer<List<Integer>> printIntegers = ints -> ints.stream()
        .map(representations::get)
        .forEach(System.out::println);

printIntegers.accept(List.of(1, 2, 3));
```

```
> Task :app:DoNotDoThis.main()
one
two
null
```

```java
Map<Integer, String> representations = new HashMap<>(ofEntries(
        entry(1, "one"),
        entry(2, "two")));

Consumer<List<Integer>> printIntegers = ints -> ints.stream()
        .map(representations::get)
        .forEach(System.out::println);

printIntegers.accept(List.of(1, 2, 3));

representations.put(3, "three");

printIntegers.accept(List.of(1, 2, 3));
```

```java
Map<Integer, String> representations = new HashMap<>(ofEntries(
        entry(1, "one"),
        entry(2, "two")));

Consumer<List<Integer>> printIntegers = ints -> ints.stream()
        .map(representations::get)
        .forEach(System.out::println);

printIntegers.accept(List.of(1, 2, 3));

representations.put(3, "three");

printIntegers.accept(List.of(1, 2, 3));
```

```
> Task :app:DoNotDoThis.main()
one
two
null
one
two
three
```

```java
Map<Integer, String> representations = new HashMap<>(ofEntries(
        entry(1, "one"),
        entry(2, "two")));

Consumer<List<Integer>> printIntegers = ints -> ints.stream()
        .map(representations::get)
        .forEach(System.out::println);

printIntegers.accept(List.of(1, 2, 3));

representations.put(3, "three");

printIntegers.accept(List.of(1, 2, 3));
```

```
> Task :app:DoNotDoThis.main()
one
two
null
one
two
three
```

- 👎 Verwendung von außerhalb definierten Variablen innerhalb der Funktionen
- 👎 Verletzung der Referentiellen Transparenz

```java
Map<Integer, String> representations = ofEntries(
        entry(1, "one"),
        entry(2, "two"));
```

```java
Map<Integer, String> representations = ofEntries(
        entry(1, "one"),
        entry(2, "two"));

Function<Map<Integer, String>, Consumer<List<Integer>>> printIntegers =
        map -> ints -> ints.stream()
                .map(map::get)
                .forEach(System.out::println);
```

```java
Map<Integer, String> representations = ofEntries(
        entry(1, "one"),
        entry(2, "two"));

Function<Map<Integer, String>, Consumer<List<Integer>>> printIntegers =
        map -> ints -> ints.stream()
                .map(map::get)
                .forEach(System.out::println);

printIntegers.apply(representations).accept(List.of(1, 2, 3));
```

```java
Map<Integer, String> representations = ofEntries(
        entry(1, "one"),
        entry(2, "two"));

Function<Map<Integer, String>, Consumer<List<Integer>>> printIntegers =
        map -> ints -> ints.stream()
                .map(map::get)
                .forEach(System.out::println);

printIntegers.apply(representations).accept(List.of(1, 2, 3));

Map<Integer, String> representationsContainingThree = ofEntries(
        entry(1, "one"),
        entry(2, "two"),
        entry(3, "three"));

printIntegers.apply(representationsContainingThree).accept(List.of(1, 2, 3));
```

```java
Function<Integer, String> representations = number -> switch (number) {
    case 1 -> "one";
    case 2 -> "two";
    default -> null;
};

Function<Function<Integer, String>, Consumer<List<Integer>>> printIntegers =
    map -> ints -> ints.stream()
        .map(map::apply)
        .forEach(System.out::println);

printIntegers.apply(representations).accept(List.of(1, 2, 3));

Function<Integer, String> oneTwoThree = number -> switch (number) {
    case 1 -> "one";
    case 2 -> "two";
    case 3 -> "three";
    default -> null;
};

printIntegers.apply(oneTwoThree).accept(List.of(1, 2, 3));
```

```java
final class ReviewedBooks {

  private final Collection<Book> books;

  ReviewedBooks(Collection<Book> books) {
       this.books = books;
  }

  Collection<Book> goodBooks() {
    return // TODO
  }
}
```

```java
interface IsAGoodBook extend Predicate<Book> {




}
```

```java
interface IsAGoodBook extends Predicate<Book> {
   IsAGoodBook IS_A_GOOD_BOOK = book -> book.authorNameStartsWith("A") &&
                                book.isTitleLength(35) &&
                                book.wasPublishedIn(2025);
}
```

```java
interface IsAGoodBook extends Predicate<Book> {
   IsAGoodBook IS_A_GOOD_BOOK = book -> book.authorNameStartsWith("A") &&
                                book.isTitleLength(35) &&
                                book.wasPublishedIn(2025);
}

@Test
void thisIsAGoodBook() {
   Book aGoodBook = new Book( new Author("Andreas Jürgensen"),
           new Title("Funktionale Programmierung mit Java"),
           new YearOfPublication(2025));

   boolean result = IS_A_GOOD_BOOK.test(aGoodBook);

   assertThat(result).isTrue();
}
```


https://github.com/FOURTEEN-IT/Funktionale-Programmierung-mit-Java

```java
interface IsAGoodBook extends Predicate<Book> {
    IsAGoodBook IS_A_GOOD_BOOK = book -> book.authorNameStartsWith("A") &&
                                         book.isTitleLength(35) &&
                                         book.wasPublishedIn(2025);
}

@Test
void thisIsAGoodBook() {
    Book aGoodBook = new Book( new Author("Andreas Jürgensen"),
            new Title("Funktionale Programmierung mit Java"),
            new YearOfPublication(2025));

    boolean result = IS_A_GOOD_BOOK.test(aGoodBook);

    assertThat(result).isTrue();
}
```

❗ Queries, keine Commands

Command-Query-Separation, Bertrand Meyer

https://github.com/FOURTEEN-IT/Funktionale-Programmierung-mit-Java

```java
final class ReviewedBooks {

  private final Collection<Book> books;

  ReviewedBooks(Collection<Book> books) {
       this.books = books;
  }

  Collection<Book> goodBooks() {
    return // TODO
  }
}
```

```java
final class ReviewedBooks {

  private final Collection<Book> books;
  private final IsAGoodBook isAGoodBook;

  ReviewedBooks(Collection<Book> books, IsAGoodBook isAGoodBook) {
       this.books = books;
       this.isAGoodBook = isAGoodBook;
  }

  Collection<Book> goodBooks() {
    return // TODO
  }
}
```

```java
final class ReviewedBooks {

  private final Collection<Book> books;
  private final IsAGoodBook isAGoodBook;

  ReviewedBooks(Collection<Book> books, IsAGoodBook isAGoodBook) {
       this.books = books;
       this.isAGoodBook = isAGoodBook;
  }

  Collection<Book> goodBooks() {
    return books.stream().filter(isAGoodBook).toList();
  }
}
```

```java
import static de.fourteen.fp.mit.java.books.IsAGoodBook.IS_A_GOOD_BOOK;

final class ReviewedBooks {

  private final Collection<Book> books;
  private final IsAGoodBook isAGoodBook;

  ReviewedBooks(Collection<Book> books, IsAGoodBook isAGoodBook) {
       this.books = books;
       this.isAGoodBook = isAGoodBook;
  }

  ReviewedBooks(Collection<Book> books) {
       this(books, IS_A_GOOD_BOOK);
  }

  Collection<Book> goodBooks() {
    return books.stream().filter(isAGoodBook).toList();
  }
}
```

- 👍 Verwendung von Interfaces
- 👍 Einbindung per Dependency Injection
- 👍 Kleine, isolierte funktionale Komponenten
- 👍 FP-Komponenten in OOP-Anwendungen

Vortrag::learnings

Durch diesen Vortrag werdet ihr lernen…

… was Funktionale Programmierung ist.

… wie mit Java funktional programmiert werden kann.

… wie Funktionale Programmierung in Java zur Performance-Optimierung eingesetzt werden kann.

… wann welches Programmierparadigma verwendet werden sollte.

… wie funktional programmierte Komponenten in objektorientierten Programme eingebunden werden sollten.

- 👍 Entwickle kleine Komponenten
- 👍 Kopple Komponenten lose miteinander
- 👍 Beschreibe Schnittstellen deklarativ
- 👍 Wähle für jede Komponente das passendste Programmierparadigma
- 👍 Kombiniere mehrere Paradigmen, aber vermische sie nicht
- 👍 Vermeide Boilerplate-Code bei der Funktionalen Programmierung mit Java
  - ↳ Interfaces als Erweiterung von `java.util.function`
  - ↳ Lambda-Ausdruck als statische Konstante im Interface

www.fourteen-it.de

info@fourteen-it.de

andreas-juergensen

github.com/FOURTEEN-IT