DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Tutorium Programmierung 4

26.05.2025

Jannes Kurzke und Fabian Bauriedl

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Inhalt

1. Lambda Ausrücke
2. Streams

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Lambda Ausdrücke

- Anonyme Funktionen, nur über Referenz ansprechbar
- Nehmen Parameter entgegen
- Verarbeiten Paramenter
- Können Wert zurückgeben

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Lambda Ausdrücke

1. parameter -> expression
2. (parameter1, parameter2) -> expression
3. (parameter1, parameter2) -> { code block }

Lambda Ausdrücke

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- **Expressions**
  - können keine Variablen anlegen
  - return keyword kann weggelassen werden
  - können keine komplexen Operationen verwendne (if, for, ...)
- **Code Blöcke**
  - können Variablen anlegen und verwalten
  - return keyword ist nötig für returns
  - Verwendugn komplexer Operationen möglich

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Lambda Beispiel

```
public static void main(String[] args) {
    LinkedList<Integer> numbers = new LinkedList<Integer>();
    for (int i = 0; i < 10; i++) {
        numbers.add(i);
    }
}
```

> Wie würdet ihr nun alle werte der Liste ausgeben lassen?

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Lambda Beispiel

```
public static void main(String[] args) {
    LinkedList<Integer> numbers = new LinkedList<Integer>();
    for (int i = 0; i < 10; i++) {
        numbers.add(i);
    }

    for (Integer number : numbers) {
        System.out.println(number);
    }
}
```

Lambda Beispiel


```java
public static void main(String[] args) {
    LinkedList<Integer> numbers = new LinkedList<Integer>();
    for (int i = 0; i < 10; i++) {
        numbers.add(i);
    }

    numbers.forEach((number) -> System.out.println(number));
}
```

Lambda Ausdruck

```java
(number) -> System.out.println(number)
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Lambda Ausdrücke

> Besteht die Lambda funktion nur aus einem Methodenaufruf, kann sie kürzer gefasst werden

- <Klasse der Methode> :: <Methode>

```
System.out::println
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Lambda Ausdrücke

```
public static void main(String[] args) {
    LinkedList<Integer> numbers = new LinkedList<Integer>();
    for (int i = 0; i < 10; i++) {
        numbers.add(i);
    }

    numbers.forEach(System.out::println);
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Vorteile von Lambdas

1. Lesbarer Code
2. Unterstützung von Funktionaler Programmierung
3. Paralele Verarbeitung mit Streams
4. Einführung neuer APIs und Bibliotheken

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Funktionale Interfaces

- Interface mit genau einer abstrakten Methode
- Nützlich um Zusammenspiel mit Lambdas und Streams
- Code einmal implementiert kann an verschidenen Stellen aufgerufen werden

Consumer Interface

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- Führt eine Aktion auf einen Eingabe Wert aus
- Liefert keinen Rückgabewert
- Abstrakte Methode: `void accept(T t)`
- Geschrieben als: `Consumer <T>`
- z.B. Verarbeitung von Listen, Logging, Debugging, etc.

```
public static void main(String[] args) {
    Consumer<String> printer = text -> System.out.println("Text: " + text);
    printer.accept("Hello World");
}
```

BiConsumer Interface

- Führt eine Aktion auf zwei Eingabe Werte aus
- Liefert keinen Rückgabewert
- Abstrakte Methode: `void accept(T t, U u)`
- Geschrieben als: `BiConsumer <T, U>`
- Praktisch für Key-Value Paare

```
public static void main(String[] args) {
    BiConsumer<String, Integer> biPrinter = (name, age) ->
        System.out.println("Name: " + name + " age: " + age);

    biPrinter.accept("Thorsten", 137);
}
```

Predicate Interface

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- Verarbeitet einen Eingabewert
- Liefert Boolean Rückgabewert
- Abstrakte Methode: `boolean test(T t)`
- Geschrieben als: `Predicate <T>`
- Praktisch für Filter-Logik

```
public static void main(String[] args) {
    Predicate<Integer> ageCheck = age -> age > 18;

    System.out.println(ageCheck.test(14));
    System.out.println(ageCheck.test(22));
}
```

Function Interface

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

- Führt eine Aktion auf einen Eingabe Wert aus
- Liefert beliebigen Rückgabewert
- Abstrakte Methode: `R apply(T t)`
- Geschrieben als: `Function<T, R>`
- Kann z.B. Typen in anderen Typen umwandeln, Längen von Strings ermitteln, etc.

```
public static void main(String[] args) {
    Function<String, Integer> lengthGetter = s -> s.length();
    System.out.println(lengthGetter.apply("HelloWorld"));
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Eigene Funktionale Interfaces

- Verwendung der `@FunctionalInterface` Annotation
- Implementierung über Interface
- Beliebige Abstrakte Funktion(en) hinzufügen

Interface Erstellen

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

```
@FunctionalInterface
interface MultiplikatorInterface {
    int calculate(int a, int b);
}
```

Interface verwenden

```
public static void main(String[] args) {
    MultiplikatorInterface multiplikator = (a, b) -> a * b;
    System.out.println("Ergebnis: " + multiplikator.calculate(10, 5));
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Vorteile eigener Funktionaler Interfaces

- Bessere Lesbarkeit und Verstädnlichkeit
- Individuelle Interface- und Methodennamen

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Javas Stream API

- Ermöglicht das Arbeiten mit Strömen
- Verarbeitung von verketteten Elementen
- Intermediäre und Terminale Operationen
- Elemente werden nicht verändert
- Verarbeitung bei Bedarf

Erzeugen von Streams

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

```
public static void main(String[] args) {
    // Felder
    int[] array = {4, 8, 15, 16, 23, 42};
    IntStream integerStream = Arrays.stream(array);

    // Datensammlung
    List<Integer> list = List.of(4, 8, 15, 16, 23, 42);
    Stream<Integer> integerStream2 = list.stream();

    // Einzelobjekte
    Stream<Integer> integerStream3 = Stream.of(4, 8, 15, 16, 23, 42);
}
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Spezifische Streams

- **IntStreams**, **DoubleStreams** und **LongStreams**
- Methoden zur verarbeitung der primitiven Datentypen

```
public static void main(String[] args) {
    int[] array = {4, 8, 15, 16, 23, 42};
    IntStream integerStream = Arrays.stream(array);
    int sum = integerStream.sum();
}
```

Intermediäre Operationen 1/3

DHBW
Duale Hochschule
Baden-Württemberg

Operation	Methode	Schnittstellen-Methode
Filtern	Stream<T> filter(predicate: Predicate<T>)	boolean test(t: T)
Abbilden	Stream<T> map(mapper: Function<T, R>)	R apply(t: T)
Abbilden	DoubleStream mapToDouble(mapper: ToDoubleFunction<T, R>)	double applyAsDouble(value: T)
Abbilden	IntStream mapToInt(mapper: ToIntFunction<T, R>)	int applyAsInt(vaue: T)

Intermediäre Operationen 2/3

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Operation	Methode	Schnittstellen-Methode
Abbilden	LongStream mapToLong(mapper: ToLongFunction<T, R>)	long applyAsLong(value: T)
Spähen	Stream<T> peek(consumer: Consumer<T>)	void accept(t: T)
Sortieren	Stream<T> sorted(comparator: Comparator<T>)	int compare(o1: T, o2: T)

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Intermediäre Operationen 3/3

Operation	Methode	Schnittstellen-Methode
Unterscheiden	Stream<T> distinct()	-
Begrenzen	Stream<T> limit(maxSize: long)	-
Überspringen	Stream<T> skip(n: long)	-

Terminale Operationen 1/2

DHBW
Duale Hochschule
Baden-Württemberg

Operation	Methode	Schnittstellen-Methode
Finden	Optional<T> findAny()	-
Finden	Optional<T> findFirst()	-
Prüfen	boolean allMatch(predicate: Predicate<T>)	boolean test(t: T)
Prüfen	boolean anyMatch(predicate: Predicate<T>)	boolean test(t: T)
Prüfen	boolean noneMatch(predicate: Predicate<T>)	boolean test(t: T)

Terminale Operationen 2/2

DHBW
Duale Hochschule
Baden-Württemberg

Operation	Methode	Schnittstellen-Methode
Aggregieren	Optional<T> min(comparator: Comparator<T>)	int compare(o1: T, o2: T)
Aggregieren	Optional<T> max(comparator: Comparator<T>)	int compare(o1: T, o2: T)
Aggregieren	long count()	-
Sammeln	R collect(collector: Collector<T, A, R>)	-
Ausführen	void forEach(action: Consumer<T>)	void accept(t: T)

Stream Beispiel

DHBW
Duale Hochschule

```
public static void main(String[] args) {

    Stream.of(4, 8, 15, 16, 23, 42).filter(
        i -> {
            System.out.println(i + ": filter 1");
            return i % 2 == 0;
        }
    )
    .filter(
        i -> {
            System.out.println(i + ": filter 2");
            return i > 15;
        }
    )
    .forEach(
        i -> System.out.println(i + ": forEach")
    );
}
```

Streams ohne Bedarfsauswertung

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

```
4: filter 1
8: filter 1
15: filter 1
16: filter 1
23: filter 1
42: filter 1
4: filter 2
8: filter 2
16: filter 2
42: filter 2
16: forEach
42: forEach
```

Streams mit Bedarfsauswertung

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

```
4: filter 1
4: filter 2
8: filter 1
8: filter 2
15: filter 1
16: filter 1
16: filter 2
16: forEach
23: filter 1
42: filter 1
42: filter 2
42: forEach
```

DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg

Unendliche Streams

- Java ermöglicht das erstellen von unendlichen Strömen
- Stream.iterate()
- Stream.generate()

Unendliche Streams Beispiel

DHBW
Duale Hochschule

```
public static void main(String[] args) {

    //Stream<T> iterate(seed: T, f: UnaryOperator<T>)
    Stream.iterate(0, i -> ++i)
        .limit(100)
        .forEach(System.out::println);

    //Stream<T> iterate(seed: T, hasNext: Predicat <T>, next: UnaryOperator<T>)
    Stream.iterate(0, i -> i < 100, i -> ++i)
        .forEach(System.out::println);

    //Stream<T> generate(s: Supplier<T>)
    Stream.generate(() -> new Random()
        .nextInt(100))
        .limit(100)
        .forEach(System.out::println);
}
```