Release von Java 8

David Burkhart, Marc Philipp

Karlsruhe, 7. April 2014

andrena
OBJECTS

Experts in agile software engineering

Intro

Demo

`SortierungDemo1-9.java`

Functional Interface

= Interface mit genau einer Methode

Beispiele:

- `Comparator`
- `Runnable`
- `Callable`
- `FileFilter`

Lambda-Ausdruck

= Anonyme Implementierung eines Functional Interface

```
Comparator<Person> comparator = new Comparator<Person>() {
        @Override
        public int compare(Person a, Person b) {
                return a.getNachname().compareTo(b.getNachname());
        }
};

Comparator<Person> comparator
    = (a, b) -> a.getNachname().compareTo(b.getNachname());
```

Lambda-Schreibweise

Parameterliste	->	Body
(Person a, Person b)	->	{ return a.getNachname().compareTo(b.getNachname()); }
(a, b)	->	{ return a.getNachname().compareTo(b.getNachname()); }
(a, b)	->	a.getNachname().compareTo(b.getNachname())
a	->	a.getNachname()
()	->	a.getNachname()

Methoden-Referenzen

Schema	Beispiel	Äquivalenter Lambda-Ausdruck
Klasse::statischeMethode	Math::abs	x -> Math.abs(x)
Klasse::instanzMethode	Person::getNachname	p -> p.getNachname()
variable::instanzMethode	einePerson::getNachname	() -> einePerson.getNachname()
Klasse::new	Person::new	() -> new Person()

Default-Methoden

= An Interface deklarierte Methode mit Default-Implementierung

Vorteil: Implementierungen müssen nicht angepasst werden

Beispiele:

- `List::sort`
- `Iterable::forEach`
- `Comparator::reversed`

@FunctionalInterface

- Annotation für Interfaces
- Compiler überprüft, ob das Interface genau eine abstrakte Methode deklariert

Compile-Fehler:

```
@FunctionalInterface
interface MyInterface {
        void firstMethod();
        void secondMethod();
}
```

Invalid '@FunctionalInterface' annotation; MyInterface is not a functional interface

Statische Methoden in Interfaces möglich

Beispiel:

```
Comparator.comparing(Person::getNachname);
```

Fragen zur Intro?

java.util.function

Standard Functional Interfaces

Vier Familien von Functional Interfaces

```
interface Function<T,R> {
  R apply(T t);
}
```

```
interface Supplier<T> {
  T get();
}
```

```
interface Predicate<T> {
  boolean test(T t);
}
```

```
interface Consumer<T> {
  void accept(T t);
}
```

Demo

`PersonFunctionalDemo1-2.java`

Vier Familien von Functional Interfaces

```
interface Function<T,R> {
  R apply(T t);
}
```

```
interface Supplier<T> {
  T get();
}
```

```
interface Predicate<T> {
  boolean test(T t);
}
```

```
interface Consumer<T> {
  void accept(T t);
}
```

```
Function<Person, String> function
                = p -> p.getNachname();
```

Vier Familien von Functional Interfaces

```
interface Function<T,R> {
  R apply(T t);
}
```

```
interface Supplier<T> {
  T get();
}
```

```
interface Predicate<T> {
  boolean test(T t);
}
```

```
interface Consumer<T> {
  void accept(T t);
}
```

```
Predicate<Person> predicate
  = p -> "Müller".equals(p.getNachname());
```

Vier Familien von Functional Interfaces

```
interface Function<T,R> {
  R apply(T t);
}
```

```
interface Supplier<T> {
  T get();
}
```

```
interface Predicate<T> {
  boolean test(T t);
}
```

```
interface Consumer<T> {
  void accept(T t);
}
```

```
Supplier<Person> supplier
  = () -> new Person("Albrecht", "Müller")
```

Vier Familien von Functional Interfaces

```
interface Function<T,R> {
  R apply(T t);
}
```

```
interface Supplier<T> {
  T get();
}
```

```
interface Predicate<T> {
  boolean test(T t);
}
```

```
interface Consumer<T> {
  void accept(T t);
}
```

```
Consumer<Person> consumer
                    = p -> personen.add(p);
```

Vier Familien von Functional Interfaces

```
interface Function<T,R> {
  R apply(T t);
}
```

```
interface Supplier<T> {
  T get();
}
```

```
interface Predicate<T> {
  boolean test(T t);
}
```

```
interface Consumer<T> {
  void accept(T t);
}
```

java.util.streams

R.I.P. For-Loop

Demo

`StreamDemo1Intro.java`

java.util.stream.Stream

Bild: www.flickr.com/photos/ryanwick/2951635488/

java.util.stream.Stream

Intermediate

- Lazy (Aufbau der Pipeline)
- Liefert Stream

Terminal

- Stößt Auswertung an
- Liefert Ergebnis

Demo

`StreamDemo2Lazyness.java`

java.util.stream.Stream

Stateless

- Jedes Element wird für sich betrachtet
- `filter`
- `map`
- `reduce`

Stateful

- Es müssen alle Elemente gesamt betrachtet werden
- `distinct`
- `sorted`

Short-Circuiting

- Nicht alle Elemente müssen betrachtet werden
- `anyMatch`
- `limit`

Demo

`StreamDemo4PipelinesShortcuts.java`

`java.util.stream.Stream`

- Sicht auf die Daten
  - vgl. `Iterator`
- Möglicherweise unbegrenzt
- Nur einmal verwendbar
- Sequentiell oder parallel

Demo

StreamDemo5Endless.java

StreamDemo3NoReuseOfStreams.java

StreamDemo6Parallel.java

StreamDemo7Parallel.java

Stream Erzeugen

- `collection.stream`
- `Stream.generate(Supplier)`
- `Stream.iterate(seed, Function)`
- `stream.parallel`
- `IntStream.range`
- `Random.ints / Random.longs / Random.doubles`
- `pattern.splitAsStream`
- `bufferedReader.lines`
- `jarFile.stream / zipFile.stream`
- `Files.walk / Files.list / Files.lines`

Suchen

- filter
- anyMatch
- allMatch
- noneMatch

Transformieren

- map
- flatMap
- reduce
- collect

Sortieren

- distinct
- sorted
- min
- max

Abkürzen

- limit
- skip

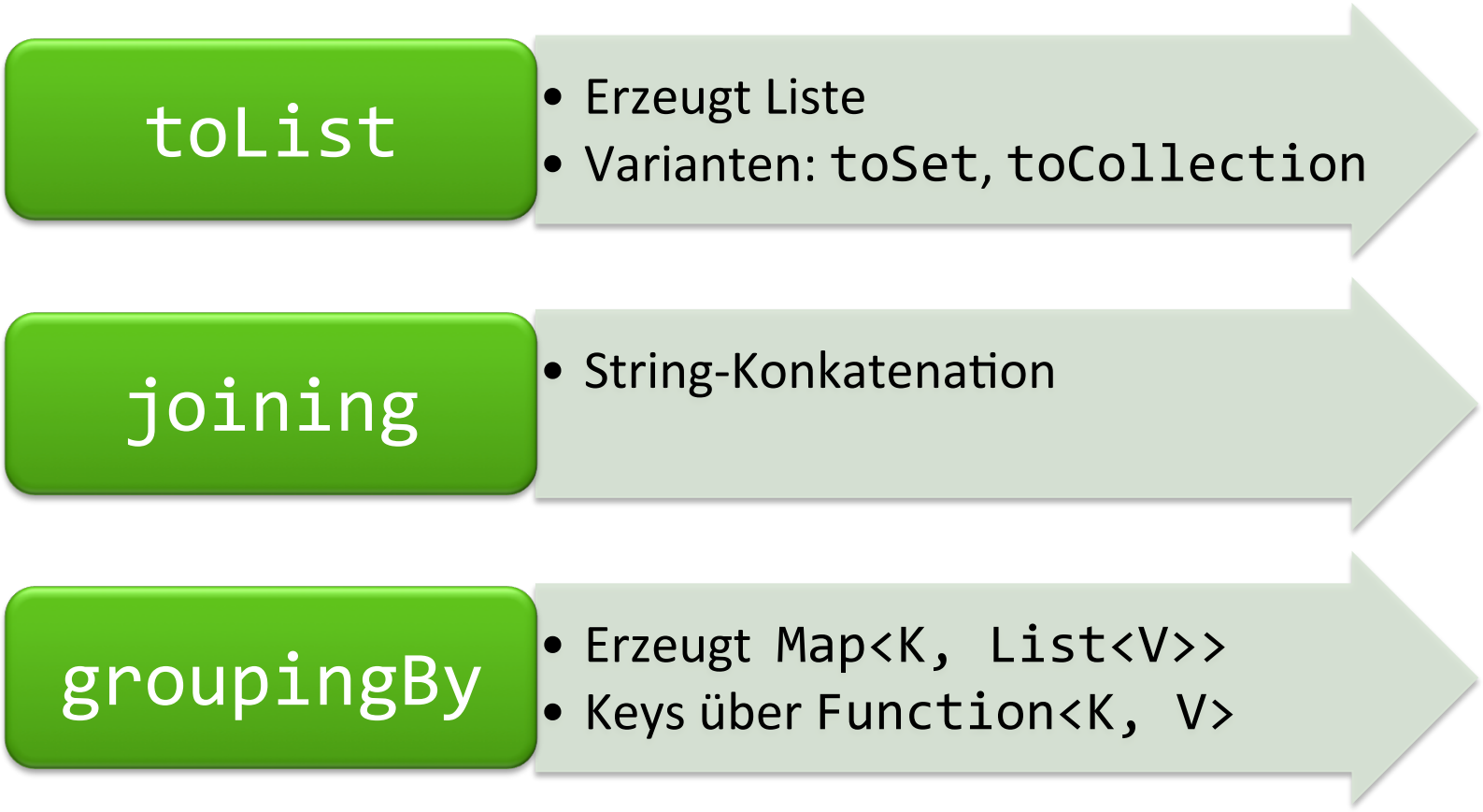
Konsumieren

- forEach
- count
- findFirst
- findAny

Primitive Streams

- mapToInt
- mapToLong
- mapToDouble

java.util.stream.Collectors



Demo

`StreamDemo9Collector.java`

Primitive Streams

Interfaces

- IntStream, LongStream, DoubleStream

SummaryStatistics

- min, max, sum, count, average

Operationen

- IntSupplier, IntConsumer, IntPredicate, IntFunction
- Long…
- Double…

java.util.Optional<T>

Error: SomeException: message
  at ...
  at ...
  at ...
  at ...
Caused by: java.lang.NullPointerException
  at ...
  at ...
  at ...
  at ...

The Billion Dollar Mistake

Tony Hoare, Erfinder von QuickSort, Turing Award Winner:

> *I call it my billion-dollar mistake. It was the invention of the null reference in 1965.*

Quelle:
http://qconlondon.com/london-2009/presentation/Null+References%3A+The+Billion+Dollar+Mistake

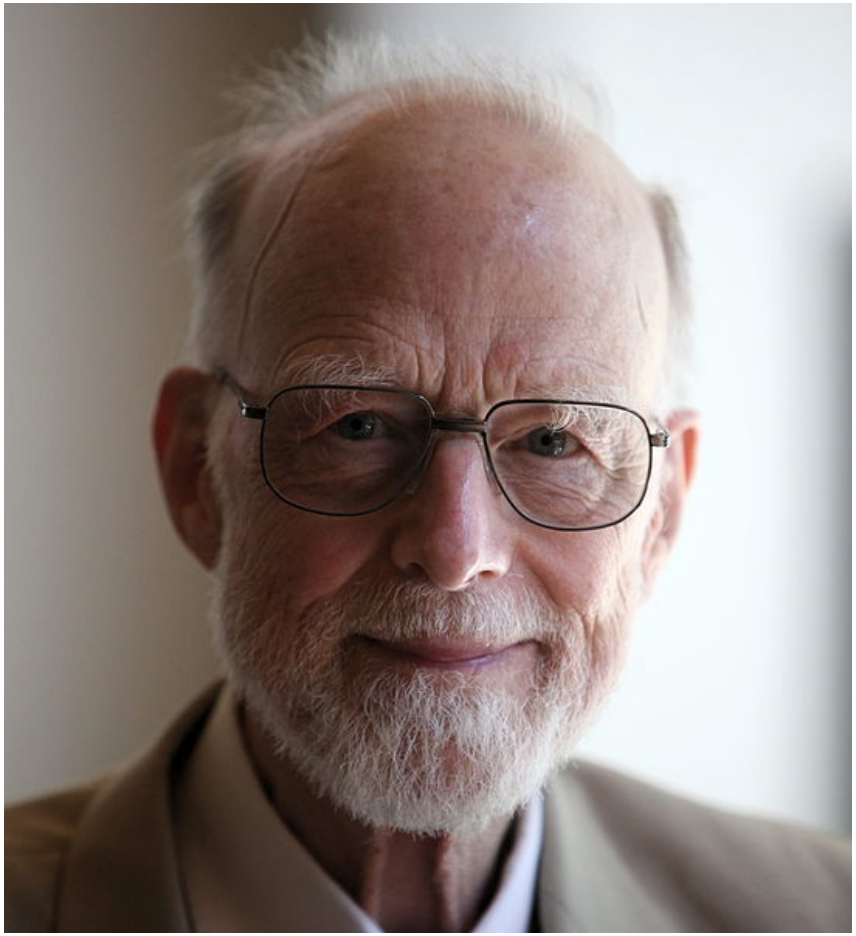


Bild: http://en.wikipedia.org/wiki/C._A._R._Hoare

Problem: Was ist die Semantik von „null“?

Was kann `map.get(key) == null` bedeuten?

Mögliche Antworten:

- Der `key` ist nicht in der `map` enthalten.
- Der `key` ist enthalten und der zugehörige Wert ist `null`.

Demo

OptionalDemo.java

java.time

Date/Time-API, Versuch #3

java.time

- Immutable
- Threadsafe (auch Formatter!)
- Brücken zur alten Welt
  - `java.util.Date`: `from(Instant)`, `toInstant`
  - `java.util.Calendar`: `toInstant`
- Warum nicht Joda-Time?
  - http://blog.joda.org/2009/11/why-jsr-310-isn-joda-time_4941.html

java.time

Maschine	Menschenlesbar	Mit Zeitzonen
• Instant	• LocalDate • LocalTime • LocalDateTime • Enum: DayOfWeek • Enum: Month • MonthDay • Year • YearMonth	• ZonedDateTime • OffsetDateTime • OffsetTime
java.util.Date		java.util.Calendar

java.time

Maschine	Menschenlesbar	Mit Zeitzonen
• Insta	• Year • YearMonth	me ime
java.util.Date		java.util.Calendar

Where possible, it is recommended to use a simpler class without a time-zone. The widespread **use of time-zones** tends to **add considerable complexity** to an application.

java.time

Duration

- Sekunden
- Nanosekunden

Period

- Jahre
- Monate
- Tage

Links

Download

- http://www.oracle.com/technetwork/java/javase/downloads/index.html

Eclipse-Support

- http://www.eclipse.org/downloads/java8/

JavaDoc

- http://download.java.net/jdk8/docs/api/java/util/function/package-summary.html
- http://download.java.net/jdk8/docs/api/java/util/stream/package-summary.html
- http://download.java.net/jdk8/docs/api/java/util/stream/Stream.html
- http://download.java.net/jdk8/docs/api/java/time/package-summary.html

Take it for a spin...

David Burkhart
Mail david.burkhart@andrena.de
andrena OBJECTS

Marc Philipp
Mail marc.philipp@andrena.de
Twitter @marcphilipp
andrena OBJECTS
Blog marcphilipp.de

Not to be confused with Lambada.

http://www.flickr.com/photos/lelik/247277980