
# JAVA IS BACK

Workshop 27.01.2015 – 29.01.2015

# EINFÜHRUNG

- Thorsten Weber
- Workshop
  - ~50 % Theorie, ~50 % Praxis
  - Spiel, Spaß und Schweiss
- Ziele:
  - schlanke, verlässliche Backend-Systeme mit Java
  - Kennenlernen von Frameworks

# EINFÜHRUNG

- Tag 1:
  - Einführung
  - Java 8 –Überblick und neue API
  - Java 8 – Lambda
  - Java 8 – Functional Data processing
  - Spring 4
  - Feature Toggles

# EINFÜHRUNG

- Tag 2:
  - Spring Jdbc
  - Spring Data und JPA
  - Spring Data und NoSQL
  - REST mit Spring

# EINFÜHRUNG

- Tag 3:
  - Messaging mit Apache Camel
  - Systemintegration mit Apache Camel
  - Microservices mit DropWizard

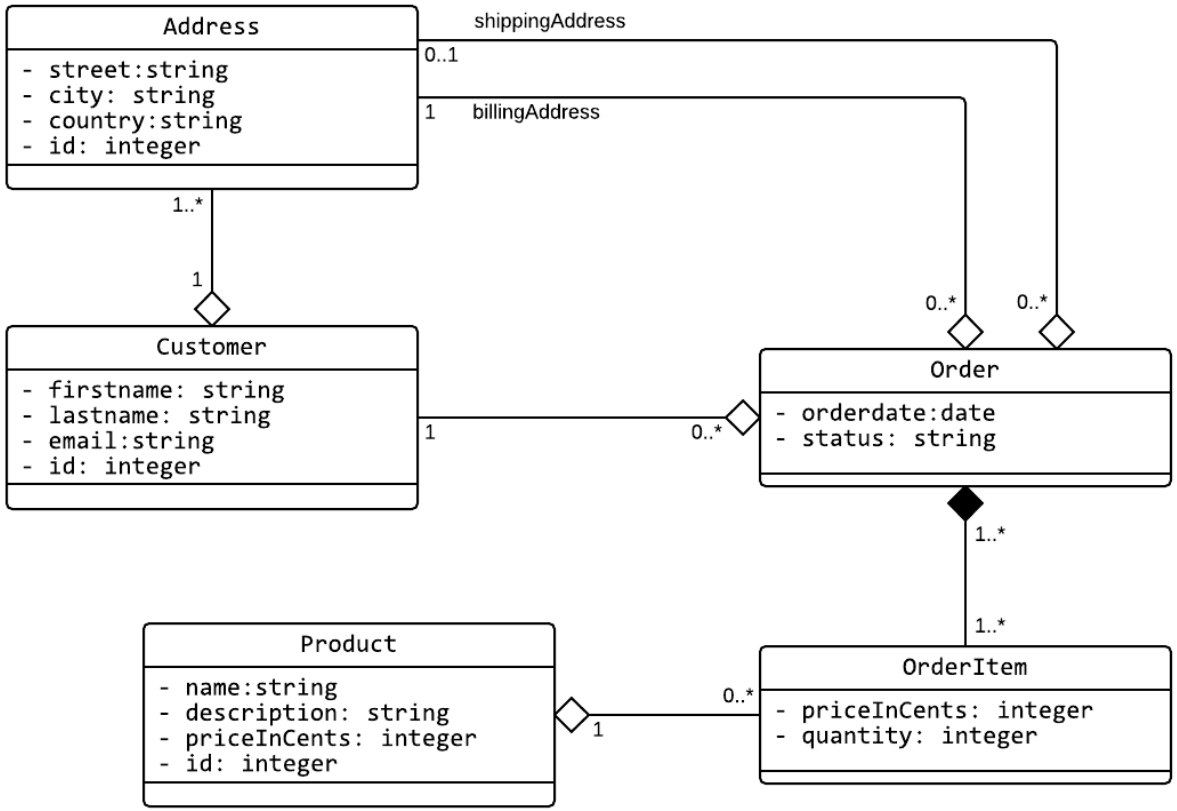
# EINFÜHRUNG

- Tools
  - Java 8 (http://java.oracle.com)
  - Maven 3 (http://maven.apache.org)
  - JUnit 4 (http://junit.org)
  - Hamcrest (http://hamcrest.org/JavaHamcrest)
  - Mockito (https://github.com/mockito/mockito)
  - PowerMockito (https://code.google.com/p/powermock)

# EINFÜHRUNG

- Domäne:

  einfacher Webshop



# EINFÜHRUNG

- Fragen!
- Mobiles im Flugmodus …
- Selbstverantwortung

# JAVA 8 – ÜBERBLICK UND NEUE API

- Java 7
  - 28.07.2011
  - Version 7u72
  - End of public updates: April 2015
- Java 8
  - 18.03.2014
  - Version 8u25

# JAVA 8 – ÜBERBLICK UND NEUE API

- `java.util.Optional`
  - Idee
    - Vermeiden von `java.lang.NullPointerException`
    - Vereinfachen von Strukturen wie `Collections`
    - Guava Google Core Libraries for Java (https://github.com/google/guava)
  - `java.util.Optional<T>`
  - `java.util.OptionalInt`
  - `java.util.OptionalLong`
  - `java.util.OptionalDouble`

# JAVA 8 – ÜBERBLICK UND NEUE API

- `java.util.Optional`
  - `Optional<String> optional = Optional.of(aString);`
  - `Optional<String> optional = Optional.ofNullable(aString);`
  - `Optional<String> optional = Optional.empty();`
  - `boolean isPresent()`
  - `T get()`
  - `T orElse(T other)`

- Analog `OptionalInt, OptionalLong, OptionalDouble`

# JAVA 8 – ÜBERBLICK UND NEUE API

- Praxis 15 min
  - Schreibt je ein kleines Programm oder einen JUnit-Test, das folgende Fragen beantwortet:
  - Was passiert, wenn für in `Optional`-Objekt `get()` aufgerufen wird, wenn `isPresent()` `false` ergibt?
  - Wie verhält sich eine Liste beim Iterieren bei `null`-Werten im Vergleich zu `Optional`-Werten?

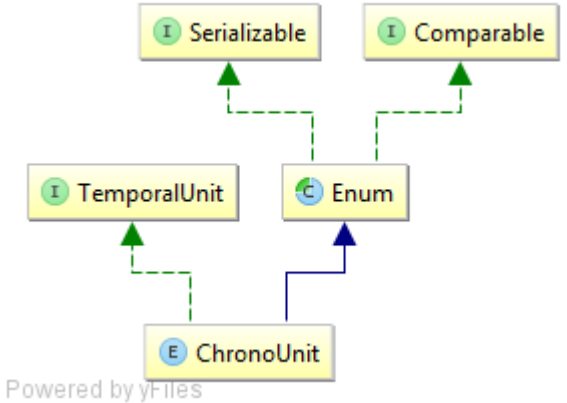
# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - http://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html
  - JSR-310 (https://jcp.org/en/jsr/detail?id=310)
  - Neue Abstraktionen für Zeit und Datum
  - Immutable und threadsafe
  - Fluent Interfaces Pattern
  - Vereinfachte Berechnungen
  - ISO-8601 Calendar System
  - Basiert auf joda-time (http://www.joda.org/joda-time)

# JAVA 8 – ÜBERBLICK UND NEUE API

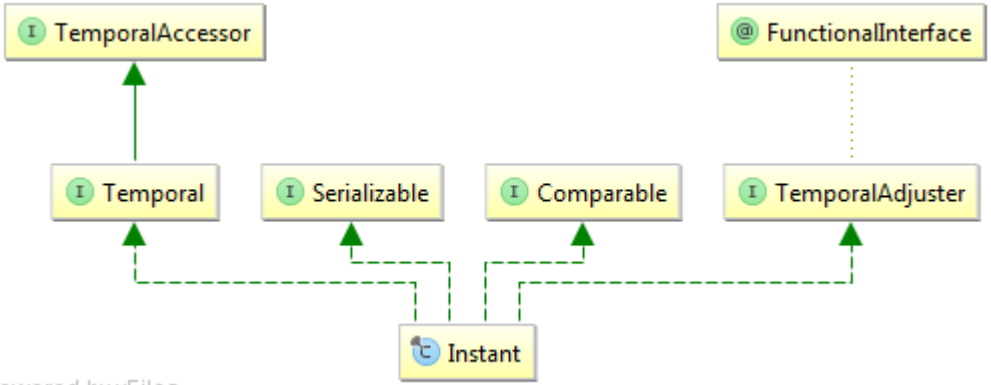
- Package `java.time.temporal`
  - `ChronoUnit` – Zeiteinheiten
  - `Enum`
  - `isTimeBased()`
  - `isDateBased()`
  - `Duration getDuration()`

  - `ChronoUnits.HOURS` – Stunden
  - `ChronoUnits.DAYS` - Tage



# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - `Instant` – Zeitpunkt
  - `now()`
  - `isBefore(Instant other)`
  - `isAfter(Instant other)`
  - `plus(TemporalAmount amount)`
  - `plus(long amountToAdd, TemporalUnit unit)`
  - `plus…`
  - `minus…`
  - `java.util.Date.toInstant()`

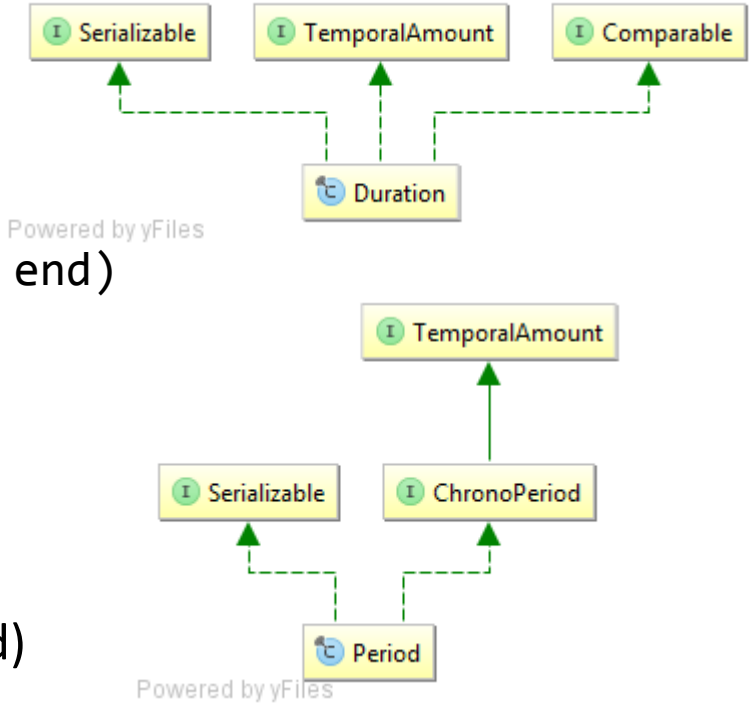


# JAVA 8 – ÜBERBLICK UND NEUE API

- Praxis 5 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, der folgende Frage beantwortet:
  - Welche Zeitpunkte sind es von jetzt an:
    - Nach 5 Stunden
    - Nach 15 Minuten
    - Dann 6 Minuten vorher

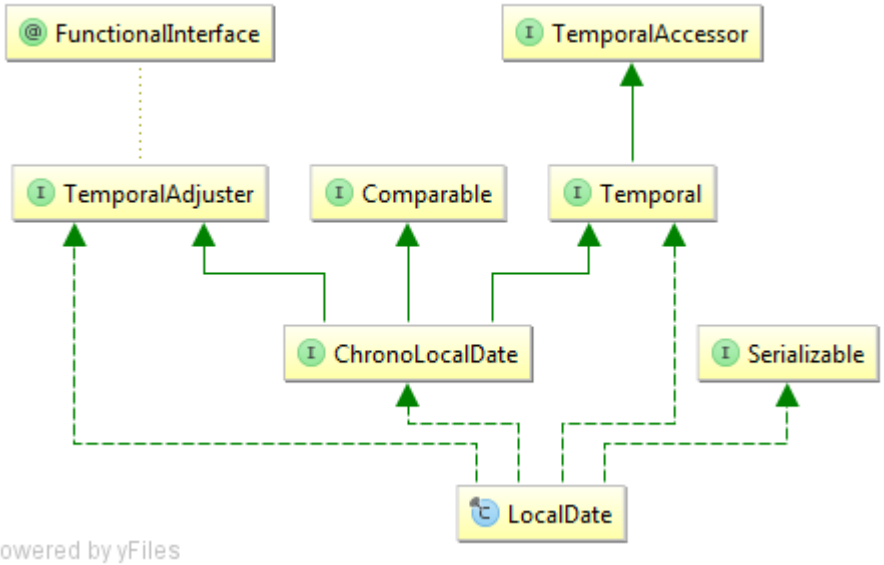
# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - `Duration` – zeitbasierte Dauer
  - `ofSeconds(long seconds)`
  - `between(Temporal start, Temporal end)`
  - `Period` – datumsbasierte Dauer
  - ofDays(int days)
  - of(long amount, TemporalUnit unit)
  - between(LocalDate start, LocalDate end)
  - YEARS, MONTHS, DAYS



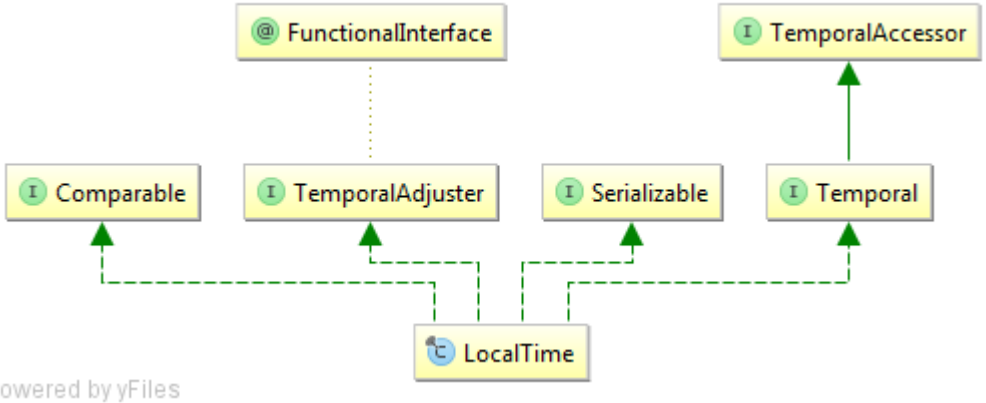
# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - `LocalDate` – Datum ohne Zeitzone
  - `now(…)`
  - `parse(…)`
  - `of(…)`
  - `with(…)`
  - `plus…`
  - `minus…`
  - `atTime(…)`



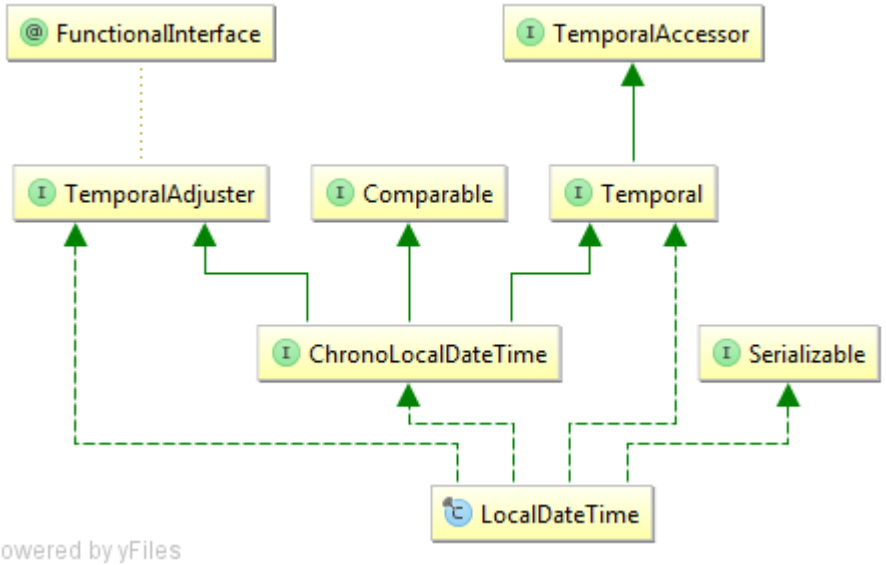
# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - `LocalTime` – Zeit ohne Zeitzone
  - `now(…)`
  - `parse(…)`
  - `of(…)`
  - `with(…)`
  - `plus…`
  - `minus…`
  - atDate(LocalDate date)



# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - `LocalDateTime` – Zeitstempel ohne Zeitzone
  - `now(…)`
  - `parse(…)`
  - `of(…)`
  - `with(…)`
  - `plus…`
  - `minus…`
  - `atZone(ZoneId zone)`



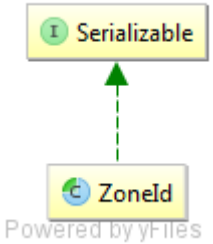
# JAVA 8 – ÜBERBLICK UND NEUE API

- Praxis 10 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, der folgende Frage beantwortet:
  - Wie alt seid Ihr:
    - In Sekunden
    - In Minuten
    - In Tagen
    - In Wochen
    - In Monaten
    - In Jahren

# JAVA 8 – ÜBERBLICK UND NEUE API

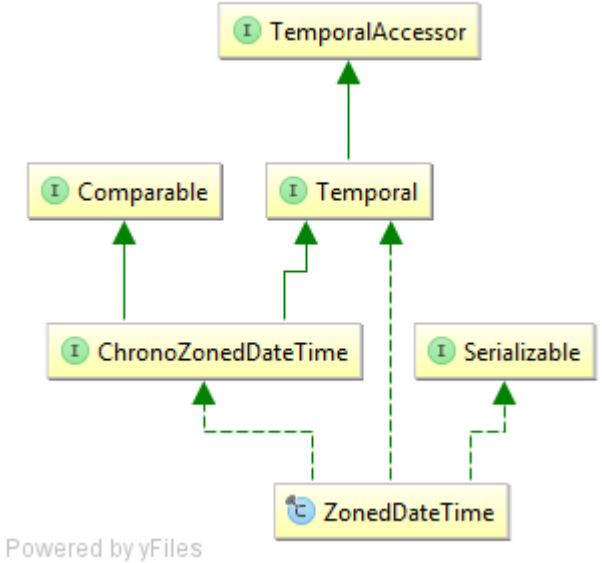
- Package `java.time`
  - `ZoneId` – Id einer Zeitzone
  - `of(String zoneId)`
  - `Set<String> getAvailableZoneIds()`
  - `systemDefault()`

  - `ZoneId.of("Europe/Berlin")`



# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - `ZonedDateTime` – Zeitstempel mit Zeitzone
  - `now(…)`
  - `parse(…)`
  - `of(…)`
  - `ofInstant(…)`
  - `with(…)`
  - `plus…`
  - `minus…`

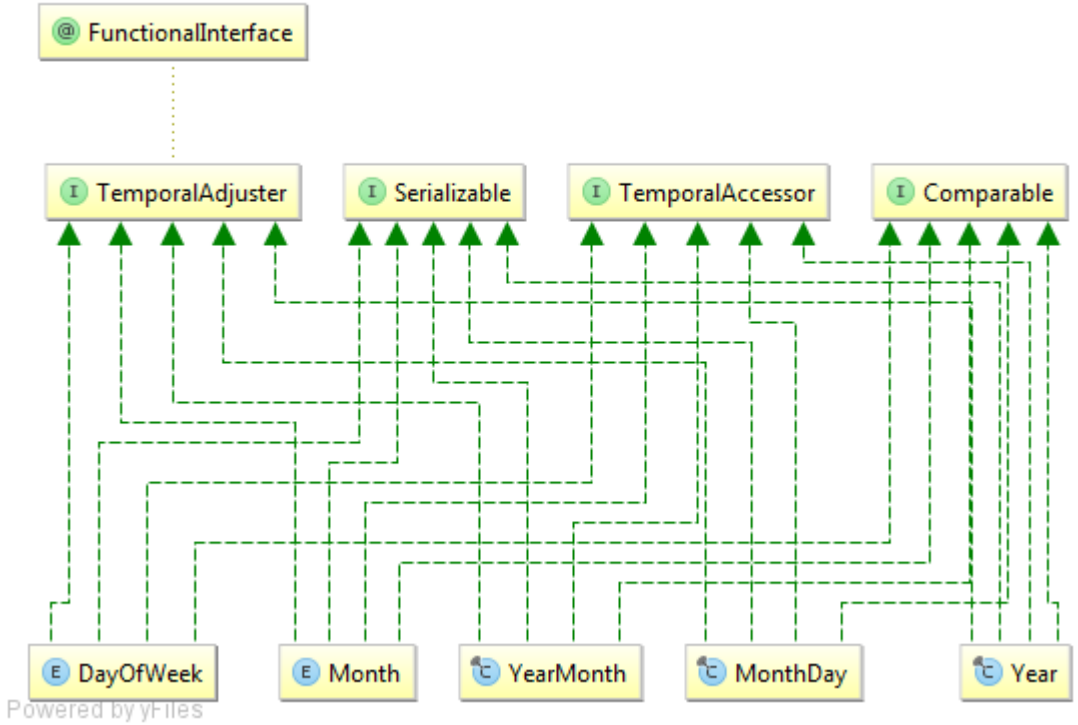


# JAVA 8 – ÜBERBLICK UND NEUE API

- Praxis 5 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, der folgende Frage beantwortet:
  - Wenn Ihr morgen um 11:15 von MUC nach Los Angeles fliegt, wieviel Uhr ist es in LA bei Eurer Ankunft nach einer Flugdauer von 12:15 Stunden?

# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time`
  - `Month – Monat`
  - `Enum`
  - `Year – Jahr`
  - `isLeap()`
  - `YearMonth – Monat-Jahr`
  - `isLeapYear`
  - `MonthDay – Tag-Monat`
  - `isValidYear(int year)`
  - `DayOfWeek – Wochentag`
  - `Enum`



# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time.format`
  - `DateTimeFormatter`
  - `parse(…)`
  - `ofPattern(…)`
  - `format(…)`
  - `ofPattern("dd.MM.yyyy HH:mm:ss")`
  - `DateTimeFormatterBuilder`

# JAVA 8 – ÜBERBLICK UND NEUE API

- Package `java.time.temporal`
  - `TemporalQuery<T> - Abfragen`
  - Interface
  - `R queryFrom(TemporalAccessor ta)`
  - `TemporalQueries mit Standard-Queries`
  - `Aufruf:`
    - `TemporalAccessor.query(TemporalQuery)`
    - `TemporalQuery.queryFrom(TemporalAccessor ta)`

# JAVA 8 – ÜBERBLICK UND NEUE API

- Praxis 10 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, das die Ausgabe der letzten Übung formatiert:
    - Für den Abflug nach deutschem Format
    - Für die Ankunft nach amerikanischem Format
  - Schreibt ein kleines Programm oder einen JUnit-Test, das für ein beliebiges Datum überprüft, ob es Euer Geburtstag ist.

# JAVA 8 – ÜBERBLICK UND NEUE API

- java.util.Base64
  - Encoding für
    - Basic (RFC 2045)
    - Url (RFC 4648)
    - MIME (RFC 2045)
  - Base64.getEncoder()
  - Base64.getDecoder()

# JAVA 8 – LAMBDA

- Funktionale Programmierung
- Groovy, Scala, C#

```
final List<String> names = Arrays.asList("1234", "1234567", "123",
        "12345678", "12");
Collections.sort(names, new Comparator<String>() {
    @Override
    public int compare(final String str1, final String str2) {
        return Integer.compare(str1.length(), str2.length());
    }
});
for (String name : names) {
    System.out.print(name.length() + ", ");
}
```

# JAVA 8 – LAMBDA

- Syntax
  - `{Parameter-Liste} -> {Ausdruck oder Anweisungen}`

  - `(int x, int y) -> { return x + y; }`
  - `(long x) -> { return x * x; }`
  - `() -> { String msg = "Lambda"; System.out.println("Hello " + msg); }`

  - `Object msg = () -> { System.out.println("1234"); }`

  Compile-Error : incompatible types: Object is not a functional interface

# JAVA 8 – LAMBDA

- Functional Interface
  - `@FunctionalInterface`
  - Nicht an Object zuweisbar!

  - Interface mit genau einer Methode (SAM-Typ Single Abstract Method)
  - Optional mit `@FunctionalInterface`-Annotation

```
@FunctionalInterface
public interface Runnable {
      void run();
}
```

# JAVA 8 – LAMBDA

- Functional Interface

```
new SAMTypeAnonymousClass() {
      public void samTypeMethod(METHOD-PARAMETERS) {
              METHOD-BODY
     }
}
```

```
{METHOD-PARAMETERS} -> { METHOD-BODY }
```

# JAVA 8 – LAMBDA

- Functional Interface

```
Runnable runnable = new Runnable() {
      @Override
      public void run() {
              System.out.println("1234");
     }
}
```

```
Runnable runnableAsLambda = () -> { System.out.println("1234"); }
```

# JAVA 8 – ÜBERBLICK UND NEUE API

- Praxis 10 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, das Strings nach ihrer Länge nach sortieren kann:
    - Mit Java 7
    - Mit Java 8

# JAVA 8 – LAMBDA

- Functional Interface
  - Typ Inference

```
(int x, int y) -> { return x + y; }
(x, y) -> x + y;
```

  - Parameter

```
Collections.sort(names, (str1, str2)->
   Integer.compare(str1.length(), str2.length());
```

  - Rückgabewert

```
public static Comparator<String> compareByLength() {
   return (s1, s2)-> Integer.compare(s1.length(), s2.length());
}
```

# JAVA 8 – LAMBDA

- Default-Methoden
  - Implementierungen von Methoden in Interfaces

```
public interface Iterable<T> {
   default void forEach(Consumer<? super T> action) {
      Objects.requireNonNull(action);
      for (T t : this) {
        action.accept(t);
      }
   }
}
```

# JAVA 8 – LAMBDA

- Default-Methoden
  - Package `java.util.function`

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);

    default Consumer<T> andThen(Consumer<? super T> after) {
        Objects.requireNonNull(after);
        return (T t) -> { accept(t); after.accept(t); };
    }
}
```

# JAVA 8 – LAMBDA

- Default-Methoden

```
interface Interface1 {
    default int method(int x) {
        return 0;
    }
}

interface Interface2 {
    default int method(int x) {
        return 1;
    }
}

class Ooops implements Interface1, Interface2 {
   …
}
```

# JAVA 8 – LAMBDA

- Default-Methoden
  - Lösung 1

```
class Ooops implements Interface1, Interface2 {
    public int method(int x) {
        return 2;
    }
}
```

  - Lösung 2:

```
class Ooops implements Interface1, Interface2 {
    public int method(int x) {
        return Interface1.super.method(x);
    }
}
```

# JAVA 8 – LAMBDA

- Methoden-Referenzen
  - Syntax Methode

    `Klasse::Methodenname - System.out::println`

  - Syntax Konstruktor

    `Klasse::new - ArrayList::new`

```
names.forEach( it -> System.out.println(it) );
names.forEach( System.out::println );
```

Aber:

```
names.forEach( it -> System.out.print(it.length() + ", "));
```

# JAVA 8 – ÜBERBLICK UND NEUE API

- Praxis 5 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, das Strings nach ihrer Länge nach sortieren kann und benutzt dazu eine Methodenreferenz.

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Funktionale Programmierung
- Bessere Parallelisierung

- Bisher:
  - Externe Iteration

```
final Iterator<String> it = names.iterator();
while (it.hasNext()) {
  final String name = it.next();
  System.out.println(name);
}
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Mit Java 8:
  - Interne Iteration

```
names.forEach(name -> System.out.println(name));

names.forEach(System.out::println);

Consumer<String> action = name -> System.out.println(name);
names.forEach(action);
```

- Erweiterung von `Iterable` um default-Methode

```
default void forEach(Consumer<? super T> action)
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Praxis 5 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, das über eine Liste von Strings iteriert und diese ausgibt:
    - Mit Java 7
    - Mit Java 8

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- `java.util.function.Predicate`:
  - @FunctionalInterface

```
boolean test(T toTest)

Predicate<String> isNull = str -> str == null;
boolean result = isNull.test("");
```

  - Default-Methoden

```
Predicate<T> negate()
Predicate<T> and(Predicate<? super T> other)
Predicate<T> or(Predicate<? super T> other)
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Praxis 10 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, das für Personen alle männlichen, über 18-jährigen herausfiltert und ausgibt:
    - Erstellt dazu eine Klasse Person mit Name, Alter und Geschlecht.
    - Benutzt `boolean Collection.removeIf(Predicate<? super E> filter)`

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- `java.util.function.UnaryOperator`:
  - @FunctionalInterface
  - extends Function<T, T>

```
static <T> UnaryOperator<T> identity()

final UnaryOperator<String> trimmer = String::trim;

void List.replaceAll(UnaryOperator<E> operator)
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Praxis 10 min
  - Schaut Euch das Package `java.util.function` unter http://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html an
  - Schreibt ein kleines Programm oder einen JUnit-Test, das für eine Liste von Namen alle in Großbuchstaben ausgibt.

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Streams
  - Folge von Verarbeitungsschritten auf Daten
  - `Quelle -> Stream -> Op1 -> … -> OpN -> Ergebnis`
  - `Create -> Intermediate -> Terminal`

- Beispiel

```
List<Person> adults = persons.stream()  <- Create
                      .filter(Person::isAdult)  <- Intermediate
                      .collect(Collections.toList());  <- Terminal
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Streams erzeugen – Create
  - Für Array oder Collections:

```
Stream<String> stream = Arrays.stream(anStringArray);
Stream<String> stream = anStringList.stream();
```

  - Sequentiell oder parallel

```
Stream<String> sequentiellStream =anStringList.stream();
Stream<String> parallelStream = anStringList.parallelStream();
Stream<String> stream = Arrays.stream(anStringArray).parallel();
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Streams erzeugen – Create
  - Vordefinierte Wertebereiche:

```
Stream<String> stream = Stream.of("1", "2", "3");
IntStream iStream = IntStream.range(1, 64);
```

  - Typen

```
Stream<T>, IntStream, DoubleStream, LongStream
```

  - Konvertierung

```
IntStream Stream.mapToInt(ToIntFunction<? super T> mapper)
LongStream IntStream.asLongStream()
Stream<Long> LongStream.boxed()
<U> Stream<U> LongStream.mapToObj(LongFunction<? extends U> mapper)
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Praxis 5 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, das eine Liste von Namen in eine Liste von Stringlängen konvertiert, multipliziert diesen Wert mit dem Faktor 0.75 und gebt das Ergebnis etwa als „Wert: 2.5„ aus.

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Operationen auf Streams – Intermediates
  - Zustandslos

```
Stream<T> filter(Predicate<? super T> predicate)
<R> Stream<R> map(Function<? super T, ? extends R> mapper)
Stream<T> peek(Consumer<? super T> action)
<R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends
R>> mapper);
```

  - Zustandsbehaftet

```
Stream<T> distinct()
Stream<T> sorted()
Stream<T> limit(long maxSize)
Stream<T> skip(long fromStart)
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Operationen auf Streams – Terminals

```
void forEach(Consumer<? super T> action)
Object[] toArray()
<R, A> R collect(Collector<? super T, A, R> collector)
Optional<T> reduce(BinaryOperator<T> accumulator)
Optional<T> min(Comparator<? super T> comparator)
Optional<T> max(Comparator<? super T> comparator)
long count()
boolean anyMatch(Predicate<? super T> predicate)
Optional<T> findFirst()
Optional<T> findAny();
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Praxis 20 min
  - Schreibt je ein kleines Programm oder einen JUnit-Test, das:
    - eine Liste von Zahlen sortiert , mehrfache herausfiltert und ausgibt
    - alle erwachsenen Personen, die mit einem M beginnen, aus einer Liste mit Personen ausgibt. Was ändert sich, wenn Ihr `peek(System.out::println)` nach jedem Schritt einfügt?
    - eine beliebige Datei mit `Files.readAllLines` einliest, alle Wörter mit weniger als 4 Buchstaben herausfiltert, Interpunktionszeichen wie ., :, ! herausfiltert, Wörter wie einer, eines, der etc. herausfiltert und eine sortierte Liste der Wörter ausgibt.

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Quiz mit reduce

```
Optional<T> reduce(BinaryOperator<T> accumulator)
```

```
final Stream<String> names = Stream.of("Mike", "Tom", "Peter", "Chris");
final Stream<Integer> integers = Stream.of(1, 2, 3, 4, 5);
final Stream<Integer> empty = Stream.of();

final Optional<String> stringConcat = names.reduce((s1, s2) -> s1 + ", " + s2);
final Optional<Integer> multiplication = integers.reduce((s1, s2) -> s1 * s2);
final Optional<Integer> addition = empty.reduce((s1, s2) -> s1 + s2);
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Quiz mit reduce

```
Optional<T> reduce(BinaryOperator<T> accumulator)
```

```
final Stream<String> names = Stream.of("Mike", "Tom", "Peter", "Chris");
final Stream<Integer> integers = Stream.of(1, 2, 3, 4, 5);
final Stream<Integer> empty = Stream.of();

final Optional<String> stringConcat = names.reduce((s1, s2) -> s1 + ", " + s2);
final Optional<Integer> multiplication = integers.reduce((s1, s2) -> s1 * s2);
final Optional<Integer> addition = empty.reduce((s1, s2) -> s1 + s2);
```

```
stringConcat: Optional[Mike, Tom, Peter, Chris]
multiplication: Optional[120]
addition: Optional.empty
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

- Utility-Klasse `java.util.stream.Collectors`

```
static <T> Collector<T, ?, Long> counting()
static <T, K> Collector<T, ?, Map<K, List<T>>> groupingBy(Function<? super T,
  ? extends K> classifier)
static Collector<CharSequence, ?, String> joining()
static <T> Collector<T, ?, Map<Boolean, List<T>>> partitioningBy(Predicate<? super
T> predicate)

final List<String> chars = Arrays.asList("11","24","999","543","333","24");
String joined = chars.stream().sorted().collect(joining("; ");
      joined : 11; 24; 24; 333; 543; 999
Map<Integer, List<String>> groupedByLength =
      names.stream().distinct().collect(groupingBy(String::length);
      groupedByLength : {2=[11,24], 3=[333, 543, 999]}
```

# JAVA 8 – FUNCTIONAL DATA PROCESSING

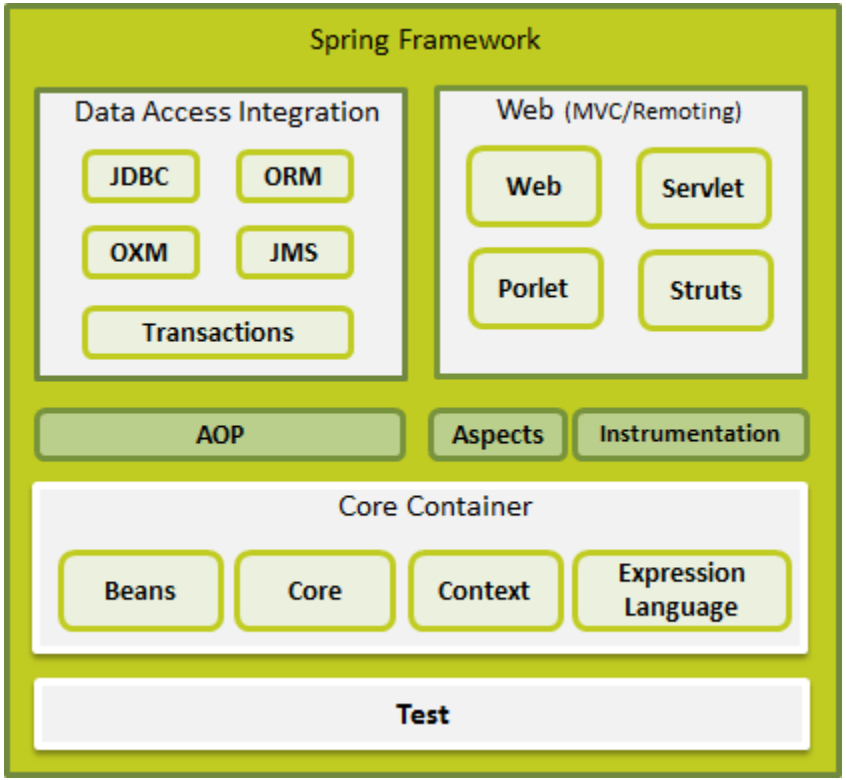
- Praxis 15 min
  - Schreibt ein kleines Programm oder einen JUnit-Test, das eine Liste von Namen
    - als mit ; getrennte, sortierte Liste ausgibt
    - nach der Länge der Wörter gruppiert ausgibt
    - nach der Länge der Wörter und deren Häufigkeit ausgibt
    - und danach partitioniert, ob ein Name ein i enthält.

# SPRING 4

- Spring http://spring.io/
- Entstanden 2002 mit  Rod Johnsons Buch *Expert One-On-One J2EE Design and Development*
- Design-Prinzipien
  - Dependency Injection
  - Aspekteorienterte Programmierung
  - Templates zur Vereinfachung
  - Programmierung nach Konvention vor Konfiguration

# SPRING 4

- Spring Boot
- Spring AMQP
- Spring Core
- Spring Dynamic Modules for OSGi Service
- Spring Extensions
- Spring Integration
- Spring BlazeDS Integration
- Spring LDAP
- Spring MVC
- Spring Rich Client
- Spring Roo
- Spring Security
- Spring Social
- Spring Web Flow
- Spring Web Services
- …



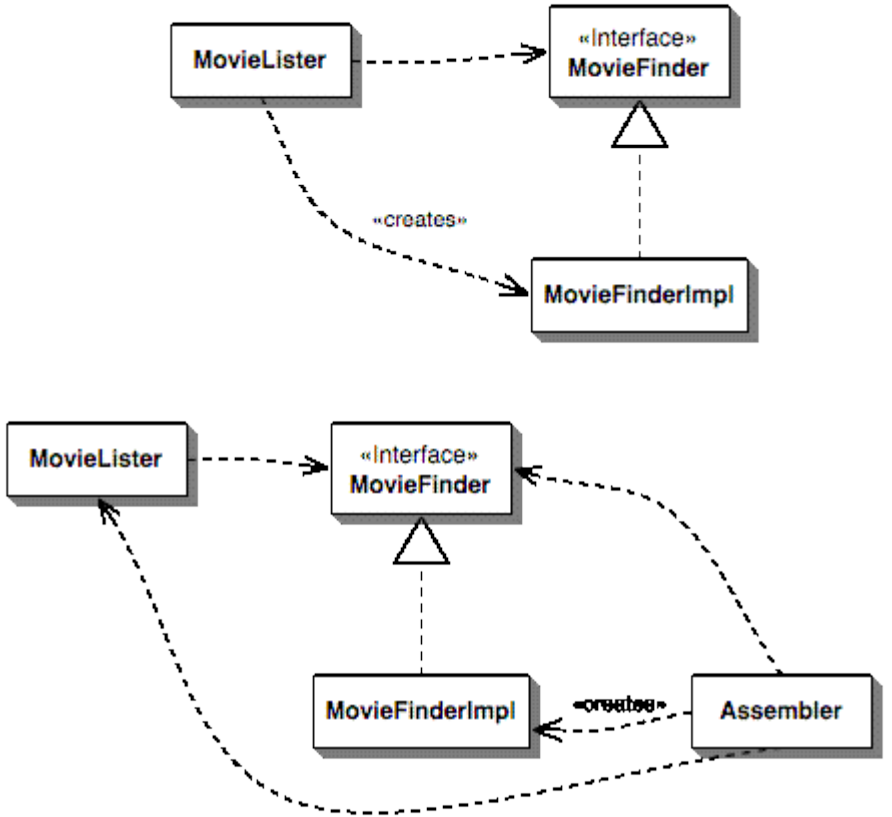
# SPRING 4

- Dependency Injection (DI)
  - 2004 von Martin Fowler eingeführt
  - Single-Responsibility-Prinzip
  - Verantwortung für Erzeugen

  und Verknüpfen von Objekten

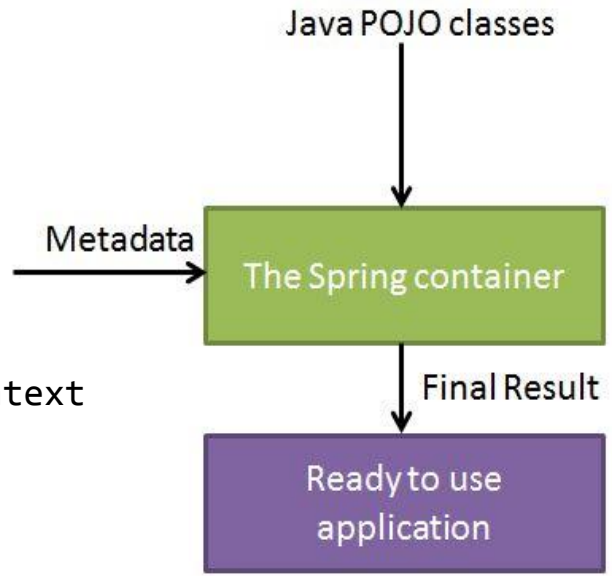
  an eigenständige Komponente

  - Constructor-based injection
  - Setter-based injection
  - Field-based injection @AutoWired



# SPRING 4

- Hello World
  - Maven-based
  - Configuration
    - Java-Annotation based @Configuration
    - XML-Based
    - `org.springframework.context.ApplicationContext`
  - Bean – von Spring verwaltetes Objekt
    - Scope per default: singleton
    - prototype
    - DI-Container



# SPRING 4

- Service-Bean
  - Servicelayer nach *Domain-Driven Design*
    - Geschäftslogik
    - Manager, typischerweise zustandslos
  - `@Service`
    - Optionaler Name `@Service("name")`
    - `@Component`
  - Annotation-basierte Konfiguration mit *Component-Scan*

```
<context:annotation-config />
<context:component-scan base-package =
            "com.jambit.workshop.jib.spring.data.jdbc" />
```

# SPRING 4

- Service-Bean
  - Servicelayer nach *Domain-Driven Design*
    - Businesslogik
  - `@Service`
    - Optionaler Name `@Service("name")`
    - `@Component`
  - Annotation-basierte Konfiguration mit *Component-Scan*

```
<context:annotation-config />
<context:component-scan base-package =
            "com.jambit.workshop.jib.spring.data.jdbc" />
```

# SPRING 4

- Praxis 30 min
  - Schreibt ein erste Spring-Anwendung:
    - Domänenmodell implementieren
    - Services für Domänenmodelle mit add, find und update
    - JUnit-Tests für alle public-Methoden
    - Ggf. mit gemockten Daten

```
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>4.1.4.RELEASE</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>4.1.4.RELEASE</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>4.1.4.RELEASE</version>
  </dependency>

  <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
  </dependency>
  <dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-core</artifactId>
      <version>1.10.19</version>
  </dependency>
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>4.1.4.RELEASE</version>
  </dependency>
```

# FEATURE TOGGLES

- Usecases
  - Continuous Integration und Continuous Delievery
  - Dark launches, Dark testing
  - Partieller Rollout
  - A/B-Testing
  - Release Date
  - Einschränkungen für User
    - via Client IP oder Usergruppe
    - via spezieller Eigenschaft

# FEATURE TOGGLES

- Togglz - http://www.togglz.org
  - Seit 2011
  - Support für Spring, JUnit, …
  - Eingebettete web-based Administrationskonsole
  - Annotations
- ff4j - http://ff4j.org
  - Seit 2013 - Feature Flipping 4 Java
  - Support für Spring

# FEATURE TOGGLES

- Togglz
  - `interface org.togglz.core.Feature`
  - Enum-Type

```
enum SpringDemoFeatures implements Feature {

  @Label("HelloWorld Feature")
  HELLOWORLD_FEATURE,

  @InDev("Release 1.2.3")
  @Label("Jira-4711")
  FEATURE_JIRA_4711;
}
```

# FEATURE TOGGLES

- Togglz
  - In Methoden

```
if(SpringDemoFeatures.HELLOWORLD_FEATURE.isActive()) {
   doWithActivatedHelloWorldFeature();
} else {
   doSomething();
}
```

  - schachtelbar
  - isActive()

```
enum SpringDemoFeatures implements Feature {
   public boolean isActive() {
        return FeatureContext.getFeatureManager().isActive(this);
   }
}
```

# FEATURE TOGGLES

- Togglz
  - Konfiguration mit `org.togglz.core.manager.TogglzConfig`

```
class SpringConfiguration implements TogglzConfig {

  public Class<? extends Feature> getFeatureClass() {
    return SpringDemoFeatures.class;
  }

  public StateRepository getStateRepository() {
    return new InMemoryStateRepository();
  }

  public UserProvider getUserProvider() {
    return () -> new SimpleFeatureUser("thorsten", true);
  }
}
```

# FEATURE TOGGLES

- Togglz
  - JUnit-Unterstützung
  - TooglzRule – Änderungen an einem Feature zur Laufzeit

```
@Rule
public TogglzRule togglzRule =
TogglzRule.allDisabled(SpringDemoFeatures.class);

togglzRule.enable(feature);
assertTrue(feature.isActive());
togglzRule.disable(feature);
```

# FEATURE TOGGLES

- Praxis 10 min
  - Schreibt eine Spring-Anwendung:
    - Gibt Hello-World aus, wenn Toggle T1 aktiviert ist, andernfalls Hi
    - Gibt Goodbye World aus, wenn Toggle T2 aktiviert ist, andernfalls Ciao
  - Mit JUnit-Tests überprüft Ihr die Funktionalität

*Zusätzlich zu den bisherigen dependencies*

```
<properties>
    <togglz.version>2.1.0.Final</togglz.version>
</properties>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-core</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-servlet</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-spring</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-console</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-testing</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-junit</artifactId>
    <version>${togglz.version}</version>
</dependency>
```

# FEATURE TOGGLES

- Togglz
  - In Spring für Klassen – Togglen von Implementierungen zur Laufzeit
  - FeatureProxyFactoryBean – Spring FactoryBean

```
<bean id="helloWorldService_V2" class="com….ImprovedHelloWorldImpl" />

<bean id="helloWorldService_V1" class="com….HelloWorldImpl" />


<bean id="helloWorldService"
class="org.togglz.spring.proxy.FeatureProxyFactoryBean">
    <property name="feature" value="FEATURE_JIRA_4711" />
    <property name="active" ref="helloWorldService_V2" />
    <property name="inactive" ref="helloWorldService_V1" />
</bean>
```

# FEATURE TOGGLES

- Togglz
  - FeatureGroup – Gruppe von Features
  - Eigene Annotation

```
@FeatureGroup
@Label("In Developement")
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface InDev {
    String value() default "";
}

@InDev("Release 1.2.3")
@Label("Jira-4712")
FEATURE_JIRA_4712
```

# FEATURE TOGGLES

- Praxis 10 min
  - Erweitert Eure Spring-Anwendung:
    - Wenn der Toggle JIRA aus der FeatureGroup InDev aktiviert ist, muss Klasse  X benutzt werden, andernfalls die bestehende.
    - Klasse X gibt Hallo Welt aus, wenn Toggle T1 aktiviert ist, andernfalls Hi
    - Gibt Auf Wiedersehen Welt aus, wenn Toggle T2 aktiviert ist, andernfalls Bye
  - Mit JUnit-Tests überprüft Ihr die Funktionalität

*Zusätzlich zu den bisherigen dependencies*

```
<properties>
    <togglz.version>2.1.0.Final</togglz.version>
</properties>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-core</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-servlet</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-spring</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-console</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-testing</artifactId>
    <version>${togglz.version}</version>
</dependency>
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-junit</artifactId>
    <version>${togglz.version}</version>
</dependency>
```

# FEATURE TOGGLES

- FF4j
  - Klasse org.ff4j.core.Feature
  - Erzeugen von Features:
    - mit XML

```
<features>
   <feature uid="HELLOWORLD_FEATURE" enable="false"
     description="HELLOWORLD_FEATURE" />
</features>
```

```
new FF4j("ff4j/ff4j.xml")
```

    - Create-Methoden

```
FF4j create(String featureName)
```

# FEATURE TOGGLES

- FF4j
  - In Methoden

```
if(ff4j.check("HELLOWORLD_FEATURE")) {
   doWithActivatedHelloWorldFeature();
} else {
   doSomething();
}
```

`feature.isEnable()` - möglich, aber weniger gut

  - schachtelbar
  - `boolean exist(String featureId)`
  - `Feature getFeature(String featureID)`

# FEATURE TOGGLES

- FF4j
  - Konfiguration

```
FF4j.setStore(FeatureStore fbs)
FF4j.setAuthorizationsManager(AuthorizationsManager am)
```

`FF4j.setAutocreate(boolean auto)` – `TFf4j.check(FEATURE)` erzeugt fehlendes Feature, default `false`

  - Default:

```
org.ff4j.store.InMemoryFeatureStore
```

# FEATURE TOGGLES

- Praxis 10 min
  - Schreibt einen JUnit-Test, der eine XML-basierte Konfiguration von FF4j testet:
    - existieren die Features?
    - sind die Features aktiviert?
    - Was passiert beim Check eines nicht-existenten Features?
    - stimmt die Description?
    - funktioniert autoCreate?

*Zusätzlich zu den bisherigen dependencies*

```
<properties>
    <ff4j.version>1.3</ff4j.version>
</properties>
<!-- ff4j -->
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-core</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-jmx</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-aop</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-test</artifactId>
    <version>${ff4j.version}</version>
    <scope>test</scope>
</dependency>
```

# FEATURE TOGGLES

- FF4j
  - Integration mit Spring
    - FF4j als Bean

```
<bean id="inMemoryFeatureStore"
class="org.ff4j.store.InMemoryFeatureStore"
    p:location="ff4j/ff4j.xml" />

<bean id="ff4jBean" class="org.ff4j.FF4j"
      p:store-ref="inMemoryFeatureStore" />
```

# FEATURE TOGGLES

- Praxis 5 min
  - Springifiziert Euren JUnit-Test.

*Zusätzlich zu den bisherigen dependencies*

```
<properties>
    <ff4j.version>1.3</ff4j.version>
</properties>
<!-- ff4j -->
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-core</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-jmx</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-aop</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-test</artifactId>
    <version>${ff4j.version}</version>
    <scope>test</scope>
</dependency>
```

# FEATURE TOGGLES

- FF4j
  - In Spring für Klassen – Togglen von Implementierungen zur Laufzeit
    - Annotation @Flip

      `name` – Name des Features

      `alterBean` – zu änderndes Bean

      `alterClazz` – Klasse der zu ändernden Bean

      `strategy` – Klasse der Toggle-Strategie z.B. `ReleaseDateFlipStrategy`
    - Einsatz

```
@Flip(name = "FEATURE_JIRA_4711", alterBean = "IMPROVED_HELLOWORLD")
String sayHello(String name)

@Component("IMPROVED_HELLOWORLD")
public class ImprovedHelloWorldImpl implements HelloWorld
```

# FEATURE TOGGLES

- Praxis 15 min
  - Schreibt einen JUnit-Test analog zu Togglz:
    - was sind die größten Unterschiede
    - sind beide equivalent?
    - welches gefällt Euch besser?

*Zusätzlich zu den bisherigen dependencies*

```
<properties>
    <ff4j.version>1.3</ff4j.version>
</properties>
<!-- ff4j -->
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-core</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-jmx</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-aop</artifactId>
    <version>${ff4j.version}</version>
</dependency>
<dependency>
    <groupId>org.ff4j</groupId>
    <artifactId>ff4j-test</artifactId>
    <version>${ff4j.version}</version>
    <scope>test</scope>
</dependency>
```

# VIELEN DANK!

WIR FREUEN UNS AUF DIE GEMEINSAME ARBEIT!

**Thorsten Weber**
thorsten.weber@jambit.com

# Bis Morgen!



jambit
WHERE INNOVATION WORKS

SOFTWARE & SYSTEM DEVELOPER
INNOVATION PARTNER
COFFEE LOVER

Sitz: München, gegründet 1999
Ziel: 100% erfolgreiche Softwareprojekte

Geschäftsführer:
Peter F. Fellinger, Markus Hartinger

Erika-Mann-Straße 63
80636 München
Tel. +49.89.45 23 47-0

office@jambit.com
www.jambit.com