ERICSSON

Java 8. Funkcjonalnie.

Ale niestandardowo.

Łukasz Krauzowicz

Kraków, 25.11.2016

O mnie

Łukasz Krauzowicz

Programista,
Miłośnik ekosystemu JVM,
Entuzjasta Otwartego Oprogramowania

lukasz.krauzowicz@ericsson.com
https://github.com/kraluk
https://github.com/ericsson

Agenda

- Dlaczego?
- Jak używasz Stream API?
- **JavaSlang**
- **cyclops-react**
- JDK 9
- Inne możliwości
- Minusy - czyli nie ma róży bez kolców

ERICSSON

Dlaczego?

ERICSSON

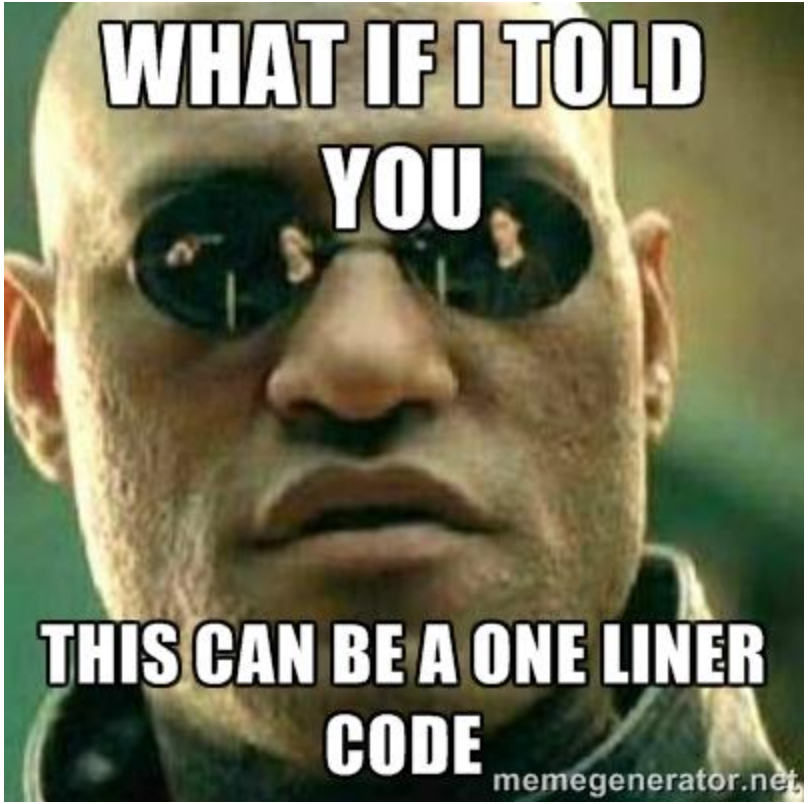
Dlaczego?

ERICSSON

- Programowanie funkcyjne
- Redundantność kodu
- Niezmienność (stałość)

Redundantnosc kodu

ERICSSON


WHAT IF I TOLD YOU
THIS CAN BE A ONE LINER CODE
memegenerator.net

Przed - Przed

ERICSSON

```
public static Dinner makeDinner(GrillService service) {
    Charcoal charcoal = service.getCharcoal();
    Lighter lighter = service.getLighter();

    if (charcoal != null && lighter != null) {
        Fire fire = service.lightFire(charcoal, lighter);
        CornCob cornCob = service.getCornCob();

        if (fire != null && corn!= null) {
            return service.grill(fire, cornCob);
        }
    }
    return null; // ???
}
```

Źródło: JavaSlang Blog

Przed

ERICSSON

```
public static Optional<Dinner> makeDinner(GrillService service) {
    Optional<Charcoal> charcoalOptional = service.getCharcoal();
    Optional<Lighter> lighterOptional = service.getLighter();

    if (charcoalOptional.isPresent() && lighterOptional.isPresent()) {
        Optional<Fire> fireOptional = service.lightFire(
                charcoalOptional.get(), lighterOptional.get());
        Optional<CornCob> cornCobOptional = service.getCornCob();

        if (fireOptional.isPresent() && cornCobOptional.isPresent()) {
            return service.grill(fireOptional.get(),
                    cornCobOptional.get());
        }
    }
    return Optional.empty();}
```

Źródło: JavaSlang Blog

Po

ERICSSON

```
public static Option<Dinner> makeDinner(GrillService service) {
    return
        For(service.getCharcoal(),
            service.getLighter(), (charcoal, lighter) -> For(
            service.lightFire(charcoal, lighter),
            service.getCornCob(), (fire, cornCob) -> For(
            service.grill(fire, cornCob)
        ).yield()));
}
```

Źródło: JavaSlang Blog

ERICSSON

Programowanie funkcyjne

Scala

ERICSSON

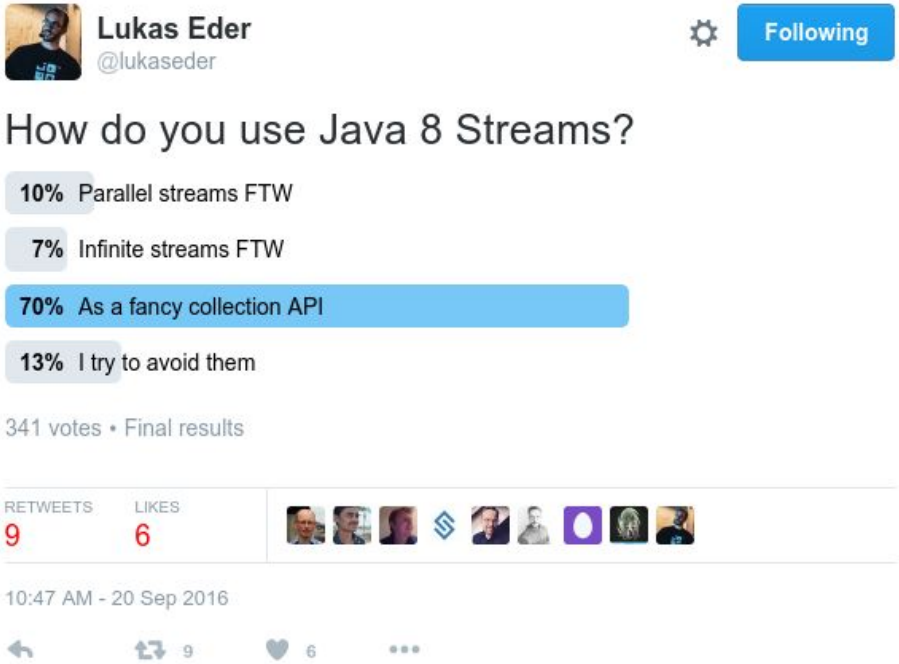
```
def makeDinner(service: GrillService): Option[Dinner] =
  for {
    charcoal <- service.getCharcoal
    lighter  <- service.getLighter
    fire     <- service.lightFire(charcoal, lighter)
    cornCob  <- service.getCornCob
    dinner   <- service.grill(fire, cornCob)
  } yield dinner
```

Źródło: JavaSlang Blog

Stream API

Jak uzywasz Stream API?

ERICSSON



ERICSSON

Dygresja

Monada

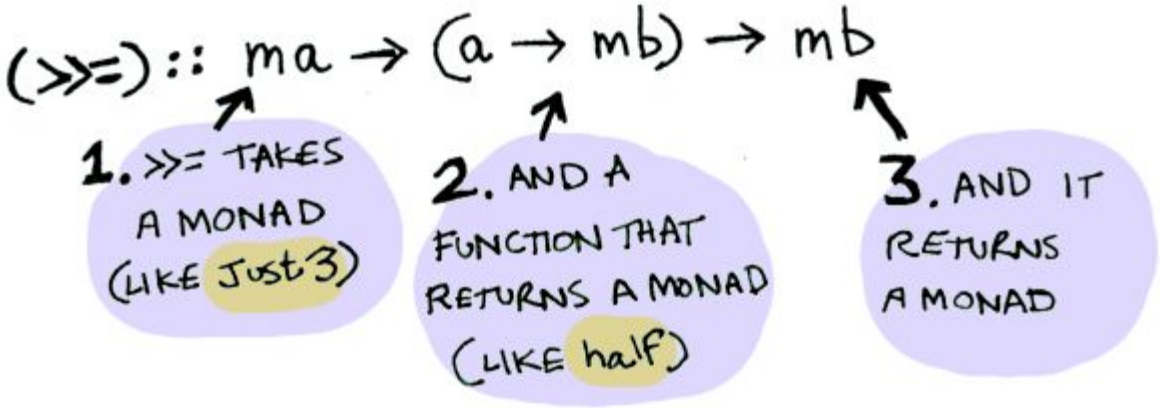
ERICSSON

What is … a monad?

Functional Programmer:
- a warm, fuzzy, little thing

Monica Monad, by FalconNL

Monada


(>>=) :: ma → (a → mb) → mb
1. >>= TAKES A MONAD (LIKE Just 3)
2. AND A FUNCTION THAT RETURNS A MONAD (LIKE half)
3. AND IT RETURNS A MONAD

ERICSSON

JavaSlang

JavaSlang

ERICSSON

- www.javaslang.io
- Wiele modułów:
  - match
  - test
  - circuitbreaker
  - render
  - jackson
- Więcej: github.com/javaslang/awesome-javaslang

ERICSSON

Option

Option

ERICSSON

```
Option<Integer> option = Option.of(someValue);
```

Optional vs Option

ERICSSON

```java
Optional<Integer> optional = Optional.of(1);

optional.get();

optional.isPresent();


Optional<Integer> empty = Optional.empty();
```

```java
Option<Integer> option = Option.of(1);

option.get();

option.isDefined();
option.isEmpty();
option.isSingleValued();

Option<Integer> none = Option.none();
Option<Void> nothing = Option.nothing();
```

ERICSSON

Pattern Matching

Pattern Matching

ERICSSON

```
if (index == 1) {
    value = "First";
} else if (index == 2) {
    value = "Second";
} else {
    value = "(empty)";
}
```

Pattern Matching (II)

ERICSSON

```
if (index == 1) {
    value = "First";
} else if (index == 2) {
    value = "Second";
} else {
    value = "(empty)";
}
```

```
String value = Match(index).of(
    Case($(1), "First"),
    Case($(2), "Second"),
    Case($(), "(empty)")
);
```

Pattern Matching (III)

ERICSSON

```
Option<String> value = Match(index).option(
    Case($(1), "First"),
    Case($(2), "Second")
);
```

Pattern Matching z Predykatem

ERICSSON

```
String value = Match(index).of(
    Case(e -> e == 1, "First"),
    Case(e -> e == 2, "Second"),
    Case($(), "(empty)")
);
```

Pattern Matching z Predykatem (II)

ERICSSON

```
String value = Match(index).of(
    Case(is(1), "First"),
    Case(isIn(2, 3), "Second or Third"),
    Case(anyOf(is(4), noneOf(is(5), is(6))), "Forth (not 5th or 6th)"),
    Case(instanceOf(BigDecimal.class), "(yolo)"),
    Case($(), "(empty)")
);
```

Pattern Matching a wartosc

ERICSSON

```
String value = Match(index).of(
    Case($(1), e -> e + ""),
    Case($(2), () -> "Second"),
    Case($(), "(empty)")
);
```

Pattern Matching a wykonanie

ERICSSON

```
Void value = Match(index).of(
    Case($(1), e -> run(() -> someWork(e))),
    Case($(2), e -> run(() -> anotherWork(e))),
    Case($(), () -> null)
);
```

Sytuacje awaryjne

Try

ERICSSON

```
Try.of(() -> someWork()).getOrElse(other);
```

Try (II)

ERICSSON

```
Result result = Try.of(this::someWork)
    .recover(e -> Match(e).of(
        Case(instanceOf(IllegalArgumentException.class), ...),
        Case(instanceOf(FileNotFoundException.class), ...),
        Case(instanceOf(NullPointerException.class), ...)
    ))
    .getOrElse(other);
```

Try (III)

ERICSSON

```
Try<Integer> success =
    Try.of(() -> 1);

Try<Integer> fail = Try.of(() -> {
    throw new Exception(
    "Something went wrong...");
});
```

```
String value = Match(success).of(
    Case(Success($()), ":-)"),
    Case(Failure($()), ";-(")
);

value =  Match(fail).of(
    Case(Success($()), ":-)"),
    Case(Failure($()), ";-(")
);
```

Try (IV)

ERICSSON

```
value = Match(success).of(
    Case(Success($(1)), "FTW!"),
    Case(Success($(e -> e == 2 )), "YOLO!"),
    Case(Failure($()), "Definitely not OK...")
);
value = Match(fail).of(
    Case(Success($()), "OK!"),
    Case(Failure($(instanceOf(RuntimeException.class))), "Very bad"),
    Case(Failure($()), "Bad")
);
```

ERICSSON

Walidacja

Validation

ERICSSON

```
public class PersonValidator {

    private static final int MIN_AGE = 18;

    public Validation<List<String>, Person> validatePerson(String name, int age) {
        return Validation.combine(validateName(name), validateAge(age)).ap(Person::new);
    }

    private Validation<String, String> validateName(String name) { // some logic... }

    private Validation<String, Integer> validateAge(int age) {
        return age < MIN_AGE
                ? Validation.invalid("Age must be greater than " + MIN_AGE)
                : Validation.valid(age);
    }
}
```

Validation (II)

ERICSSON

```
PersonValidator personValidator = new PersonValidator();

// Valid(Person(Leszke Smieszke, 42))
Validation<List<String>, Person> valid = personValidator
    .validatePerson("Leszke Smieszke", 42);

// Invalid(List(Age must be greater than 18))
Validation<List<String>, Person> invalid = personValidator
    .validatePerson("Justynian Bimber", -1);
```

Leniwe operowanie

Lazy

ERICSSON

```
Lazy<Double> lazy = Lazy.of(Math::random);
lazy.isEvaluated(); // = false
lazy.get();         // = 0.123 (random generated)
lazy.isEvaluated(); // = true
lazy.get();         // = 0.123 (memoized)
```

Lazy (II)

ERICSSON

```
CharSequence value = Lazy.val(() -> "RTM!", CharSequence .class);
```

ERICSSON

Rozszerzenia Collections Framework

Lists

ERICSSON

```
java.util.Arrays.asList(14, 42, 99)
    .stream()
    .reduce((i, j) -> i + j);
```

```
javaslang.collection.List
    .of(14, 42, 99)
    .sum();
```

```
java.util.stream.IntStream
    .of(14, 42, 99)
    .sum();
```

Lists (II)

ERICSSON

```
Stream.of(14, 42, 99)
  .sorted()
  .collect(Collectors.toList());

IntStream.of(14, 42, 99)
  .sorted()
  .collect(ArrayList::new, List::add,
        List::addAll);

IntStream.of(14, 42, 99)
  .sorted()
  .boxed()
  .collect(Collectors.toList());
```

```
javaslang.collection.List
    .of(14, 42, 99)
    .sort();
```

ERICSSON

Rozszerzenia Stream API

Streams

ERICSSON

```
javaslang.collection.Stream.from(1).filter(i -> i % 2 == 0);

for (double random : Stream.continually(Math::random).take(666)) {
    // some logic...
}
```

Streams (II)

ERICSSON

```
interface javaslang.collection.Traversable<T>
```

Uwaga!

ERICSSON

```
java.util.List != javaslang.collection.List
```

```
java.util.stream.Stream != javaslang.collection.Stream
```

Immutability!

Konwersja do standardu

ERICSSON

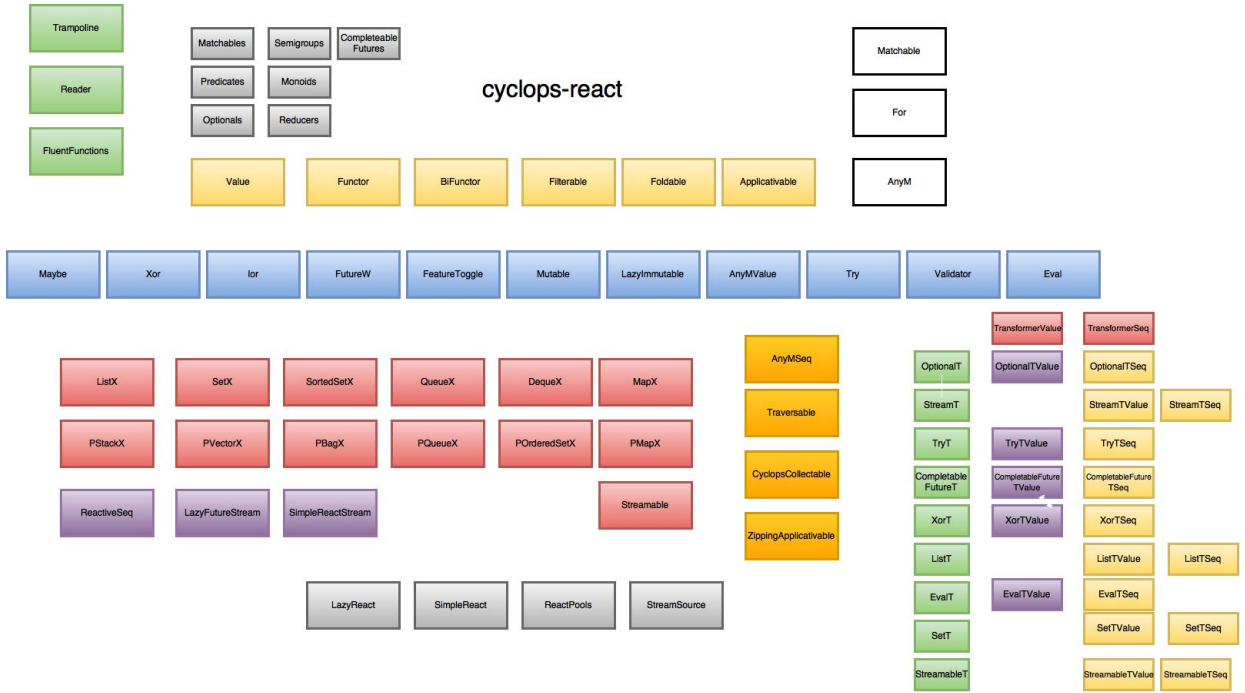
```
List.of(14, 42, 99).toJavaList();
List.of(14, 42, 99).toJavaCollection(ArrayList::new);
List.of(14, 42, 99).toJavaArray();
List.of(14, 42, 99).toJavaArray(Integer.class);

Stream.of(14, 42, 99).toJavaStream();
```

cyclops-react

cyclops-react

- github.com/aol/cyclops-react



Integracja z JavaSlang

Integracja z JavaSlang

ERICSSON

- moduł **cyclops-javaslang**
- wsparcie reactive-streams dla wszystkich typów JavaSlang
- wsparcie dla cache'owania funkcji JavaSlang
- konwertery
- dekoratory

```
Javaslang.tryM(Try.of(this::success))
        .map(String::toUpperCase)
        .toList();
```

ERICSSON

Rozszerzenia Collections Framework

Kolekcje

`ListX`

`DequeX`

`MapX`

`QueueX`

`SetX`

`SortedSetX`

`CollectionX`

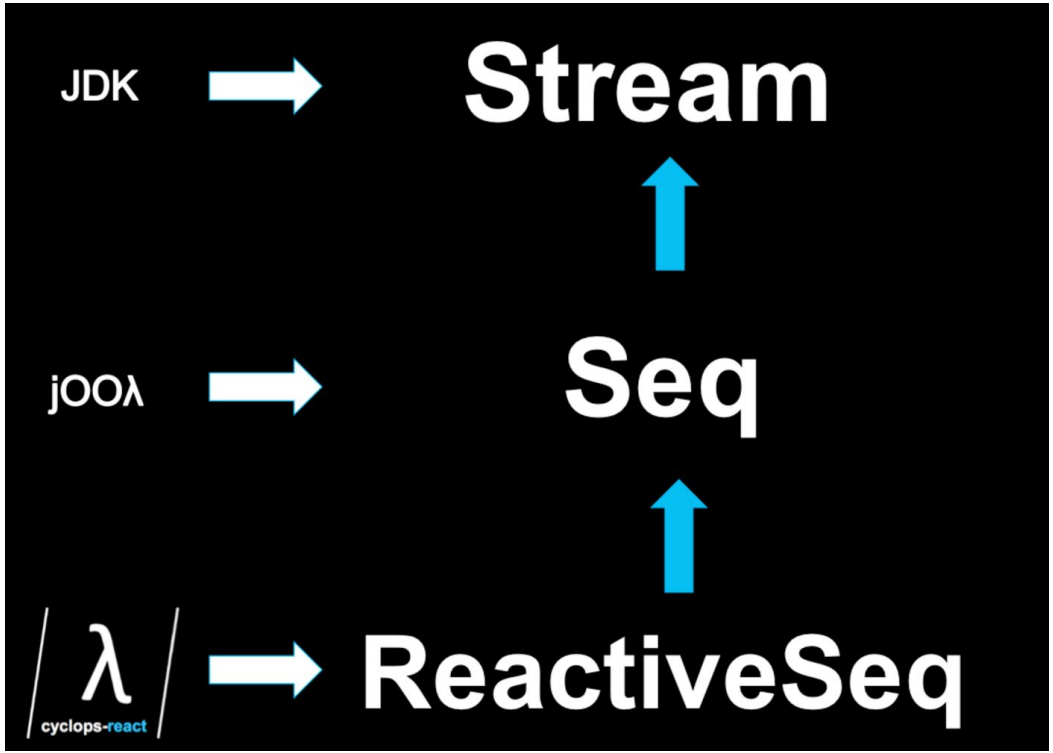
- nazwa
- implementują interfejsy z JDK

ERICSSON

Rozszerzenia Stream API

ReactiveSeq

ERICSSON



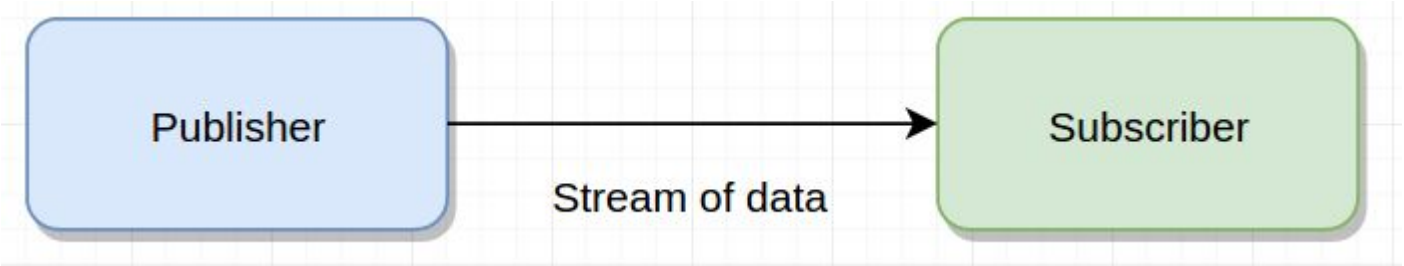
JDK 9

Flow API

Flow API

ERICSSON

- Flow API == Reactive Streams Specification
  - www.reactive-streams.org
  - wersja 1.0.0
- Reactive Programming
- Dokumentacja dostępna w API JDK 9



Flow API (II)

ERICSSON

```
@FunctionalInterface
public static interface Publisher<T> {
    // (...)
}

public static interface Subscriber<T> {
    // (...)
}

public static interface Subscription {
    // (...)
}

public static interface Processor<T, R>  extends Subscriber<T>, Publisher<R> {
}
```

Inne mozliwosci

Inne mozliwosci

ERICSSON

- jOOL — github.com/jOOQ/jOOL
- streamex — github.com/amaembo/streamex
- protonpack — github.com/poetix/protonpack
- Więcej: — github.com/akullpp/awesome-java

GitHub

Plusy

ERICSSON

Jakie sa, kazdy widzi :-)

ERICSSON

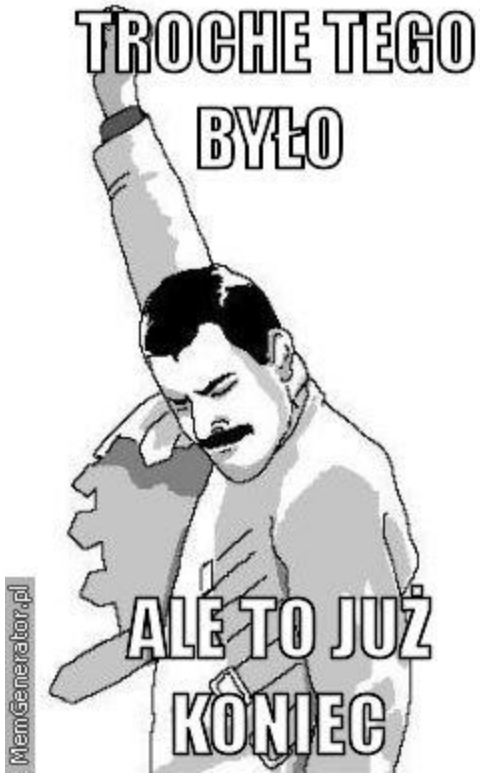
Minusy

Minusy (zawsze musza byc...)

ERICSSON

- wysoki próg wejścia
- debugging (ale czy potrzebny?)
- utrzymanie
- a jak moda przeminie?

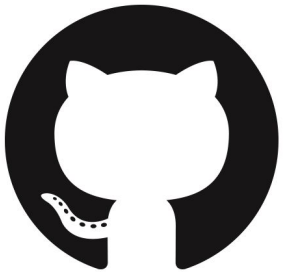
ERICSSON



ERICSSON

Materialy

ERICSSON



github.com/kraluk/functional-java-workshop

Dzieki za uwage!