LAMBDA expr3ssions

Rikhard Goldenberg
SberTech JavaSchool, Feb.2018

СБЕРБАНК ТЕХНОЛОГИИ

LAMBDA expressions

How did we get them?

JAVA SE 7

2011

JSR-292

JAVA SE 7

2011

JSR-292

“INVOKEDYNAMIC”

INVOKEDYNAMIC

- **Added to facilitate *dynamic language* development**
- **Interpreted at runtime, akin to “duck typing”**
- **Associates a *bootstrap method* with a method call**

INVOKEDYNAMIC

- **Added to facilitate *dynamic language* development**
- **Interpreted at runtime, akin to "duck typing"**
- **Associates a *bootstrap method* with a method call**

2014
JAVA 8

Behavior parameterization

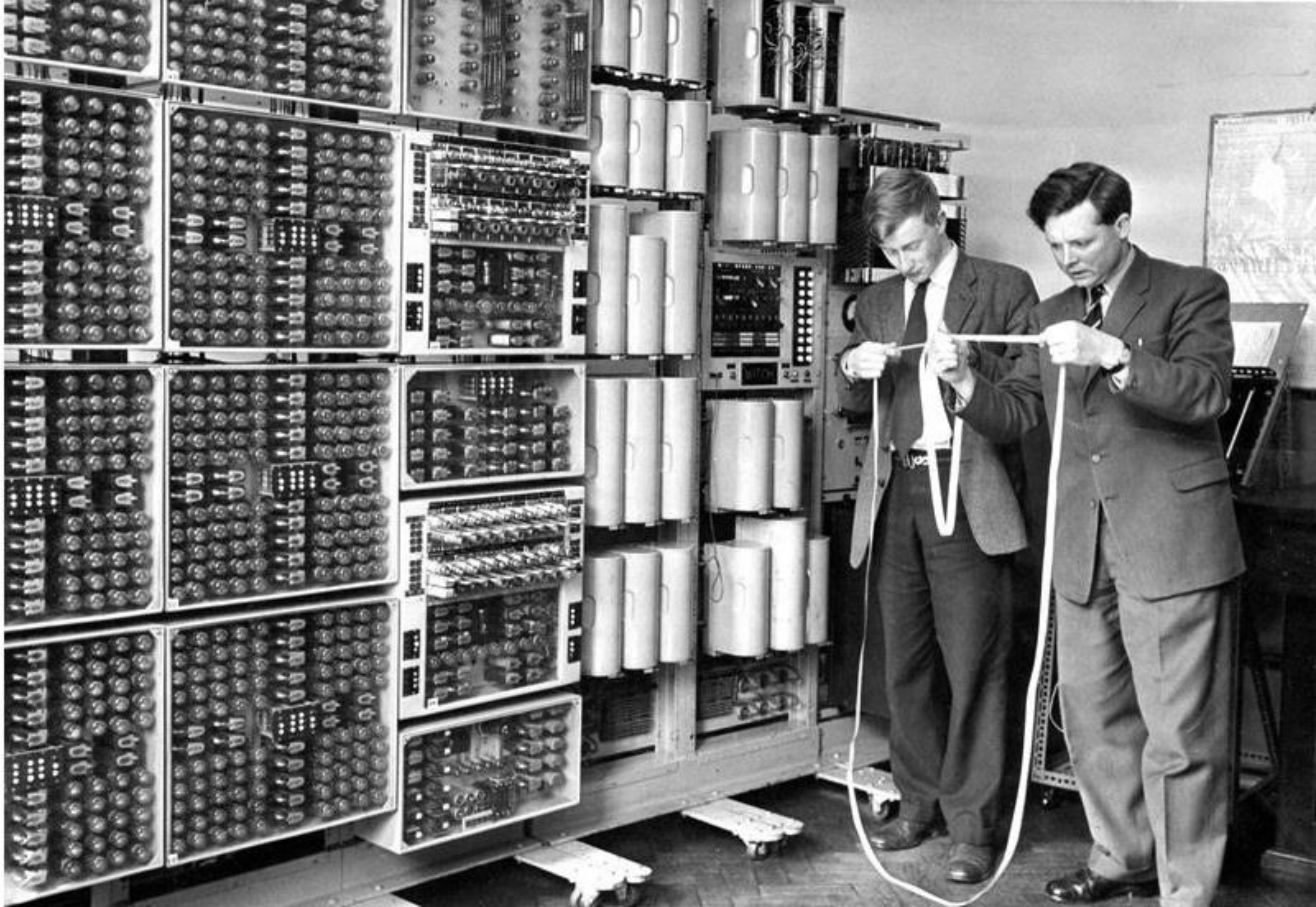
- *passing* program behavior *around*
- referencing a code fragment without immediately executing it

Behavior parameterization

- *passing* program behavior *around*
- referencing a code fragment without immediately executing it

```
public static List<Apple> filterApples(List<Apple> inventory, ????);
```

PRIOR TO JAVA 8



PRIOR TO JAVA 8

```
public interface ApplePredicate {
    boolean test(Apple apple);
}
```

PRIOR TO JAVA 8

```java
public interface ApplePredicate {
    boolean test(Apple apple);
}
```

```java
public static List<Apple> filterApples(List<Apple> inventory, ApplePredicate p) {
    List<Apple> result = new ArrayList<>();
    for (Apple apple : inventory) {
        if (p.test(apple)) {
            result.add(apple);
        }
    }
    return result;
}
```

PRIOR TO JAVA 8

```java
public class GreenApplePredicate implements ApplePredicate {
    @Override
    public boolean test(Apple apple) {
        return "green".equals(apple.getColor());
    }
}
public class HeavyApplePredicate implements ApplePredicate {
    private static final int MAX_WEIGHT = 150;

    @Override
    public boolean test(Apple apple) {
        return apple.getWeight() > MAX_WEIGHT;
    }
}
```

PRIOR TO JAVA 8

```
List<Apple> redAndHeavyApples =
            filterApples(inventory, new HeavyApplePredicate());


List<Apple> redAndHeavyApples =
            filterApples(inventory, new GreenApplePredicate());
```

СБЕРБАНК ТЕХНОЛОГИИ

PRIOR TO JAVA 8

```
List<Apple> redAndHeavyApples =
            filterApples(inventory, new HeavyApplePredicate());


List<Apple> redAndHeavyApples =
            filterApples(inventory, new GreenApplePredicate());
```

VERBOSE, LOTS OF EXTRA CODE TO MAINTAIN

Smoke weed everyday
Ingest cannabis on a daily basis
Ignite a plant containing THC using the appropriate tool on every day of the Gregorian calendar year with the presumed goal of achieving 'highness'

VERBOSITY

СБЕРБАНК ТЕХНОЛОГИИ

PRIOR TO JAVA 8

```
List<Apple> redApples = filterApples(inventory, new ApplePredicate() {
    @Override
    public boolean test(Apple apple) {
        return "green".equals(apple.getColor());
    }
});

List<Apple> heavyApples = filterApples(inventory, new ApplePredicate() {
    @Override
    public boolean test(Apple apple) {
        return apple.getWeight() > 150;
    }
});
```

СБЕРБАНК ТЕХНОЛОГИИ

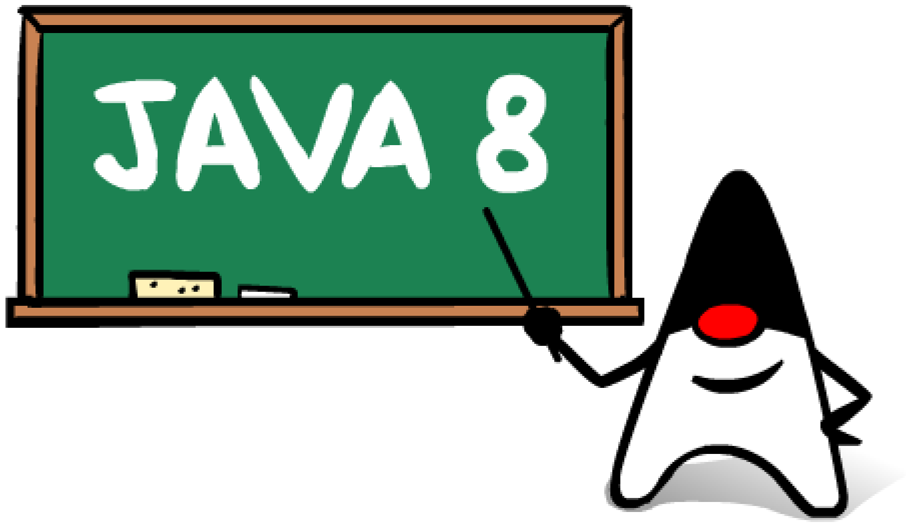
PRIOR TO JAVA 8

```
List<Apple> redApples = filterApples(inventory, new ApplePredicate() {
    @Override
    public boolean test(Apple apple) {
        return "green".equals(apple.getColor());
    }
});

List<Apple> heavyApples = filterApples(inventory, new ApplePredicate() {
    @Override
    public boolean test(Apple apple) {
        return apple.getWeight() > 150;
    }
});
```

BETTER, BUT STILL HAS LOTS OF CLUTTER

СБЕРБАНК ТЕХНОЛОГИИ

```
List<Apple> redApples
        = filterApples(inventory, apple -> "green".equals(a.getColor()));

List<Apple> heavyApples
        = filterApples(inventory, apple -> a.getWeight() > 150);
```



СБЕРБАНК ТЕХНОЛОГИИ

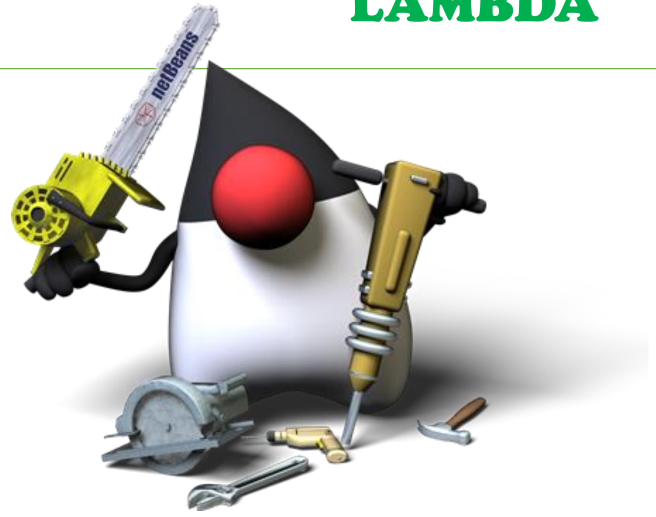
WHEN AND WHERE CAN WE USE LAMBDAS?

WHEN AND WHERE CAN WE USE LAMBDAS?

ENTER FUNCTIONAL INTERFACES!

LAMBDA

СБЕРБАНК ТЕХНОЛОГИИ



Functional Interface	Parameter Types	Return Type	Abstract Method Name	Description
Runnable	none	void	run	Runs an action without arguments or return value
Supplier<T>	none	T	get	Supplies a value of type T
Consumer<T>	T	void	accept	Consumes a value of type T
BiConsumer<T, U>	T, U	void	accept	Consumes values of types T and U
Function<T, R>	T	R	apply	A function with argument of type T
BiFunction<T, U, R>	T, U	R	apply	A function with arguments of types T and U
UnaryOperator<T>	T	T	apply	A unary operator on the type T
BinaryOperator<T>	T, T	T	apply	A binary operator on the type T
Predicate<T>	T	boolean	test	A Boolean-valued function
BiPredicate<T, U>	T, U	boolean	test	A Boolean-valued function with two arguments

FUNCTION DESCRIPTOR

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

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

FUNCTION DESCRIPTOR

```
@FunctionalInterface
public interface Predicate<T> {            T -> boolean
    boolean test(T t);
}

@FunctionalInterface
public interface Function<T, R> {             T -> R
    R apply(T t);
}
```

METHOD REFERENCES

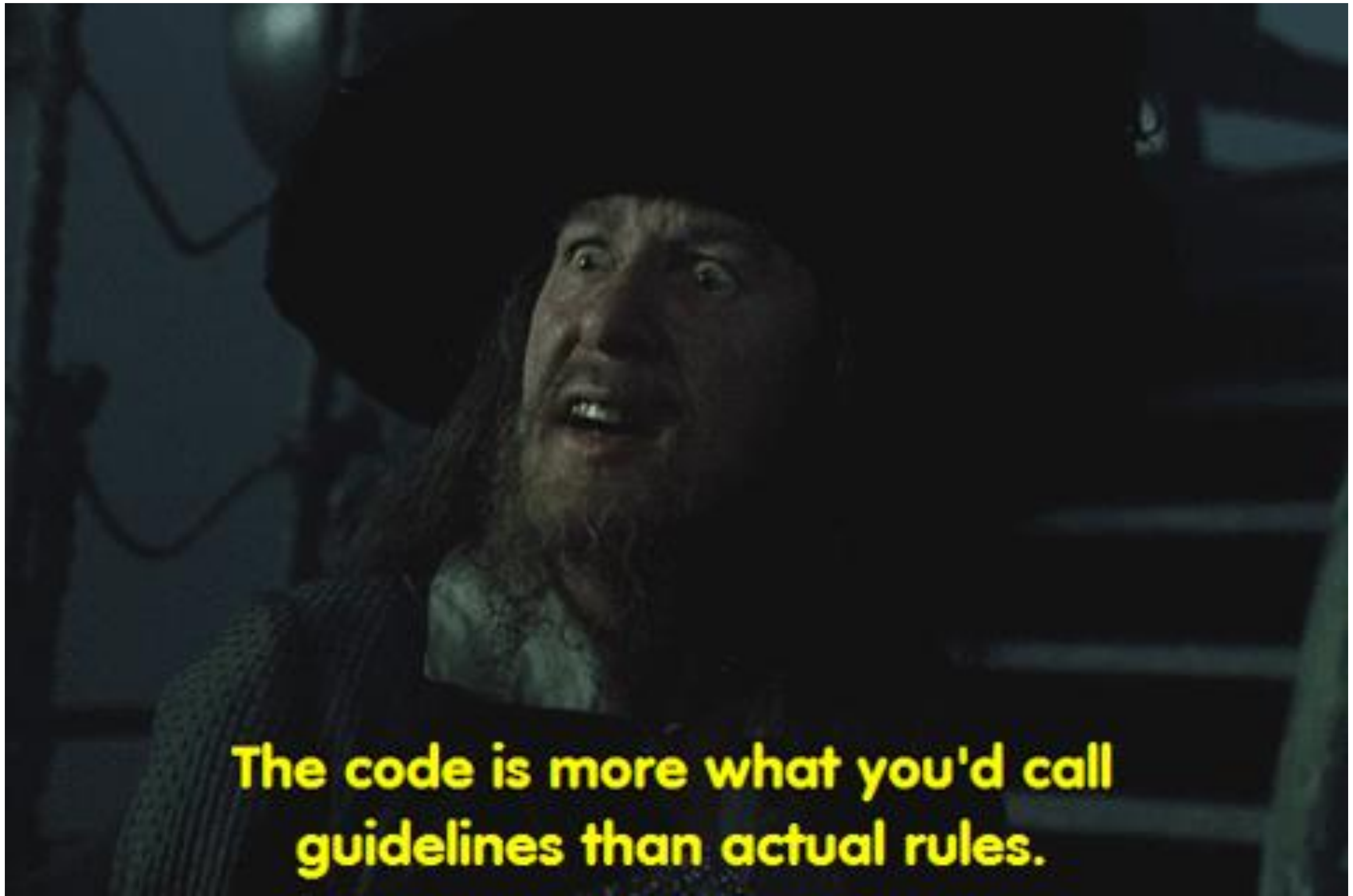
```
Collections.sort(strings, (first, second) -> first.compareToIgnoreCase(second));
```

…becomes…

```
Collections.sort(strings, String::compareToIgnoreCase)
```

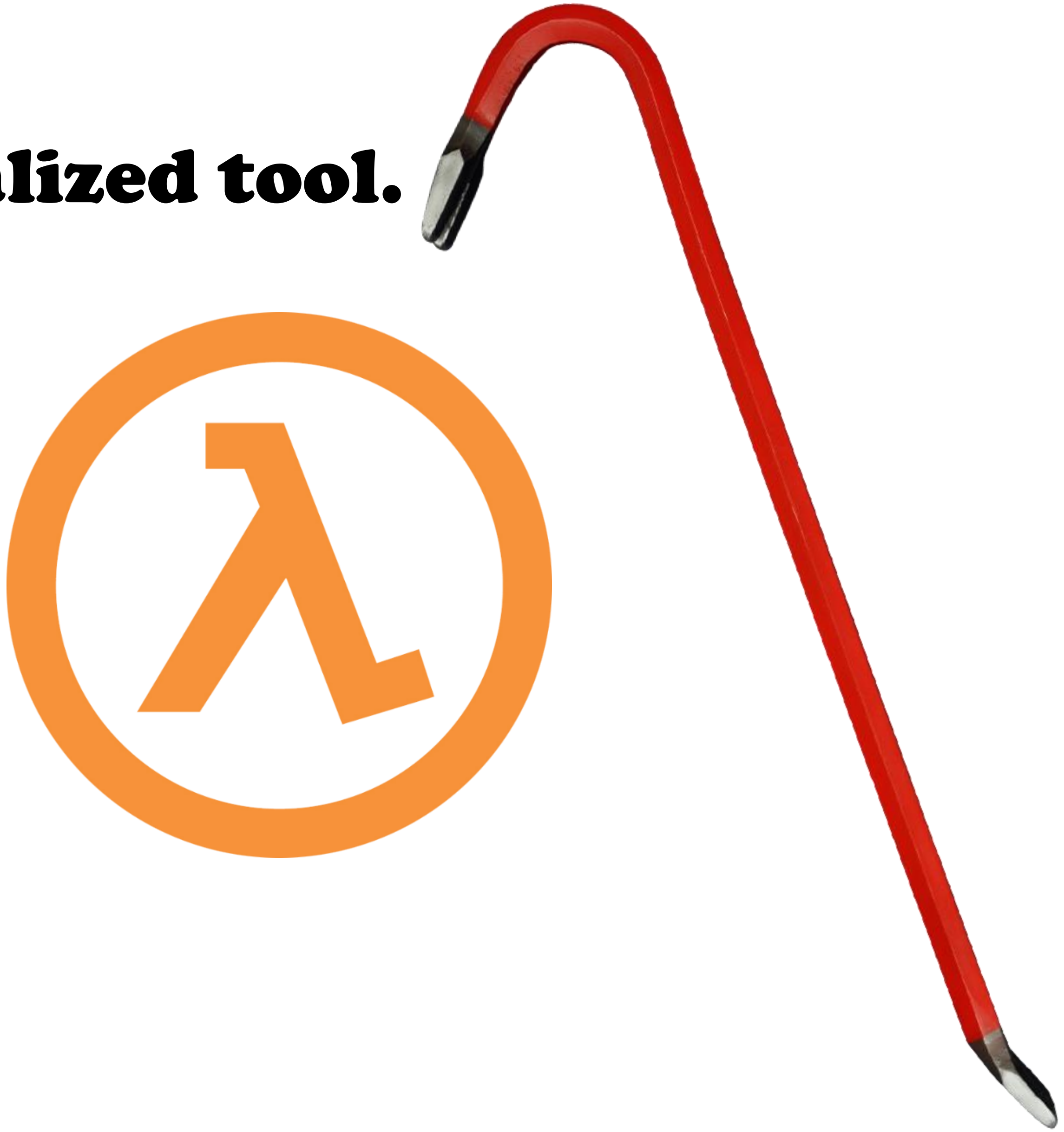
JAVA LAMBDA RULES

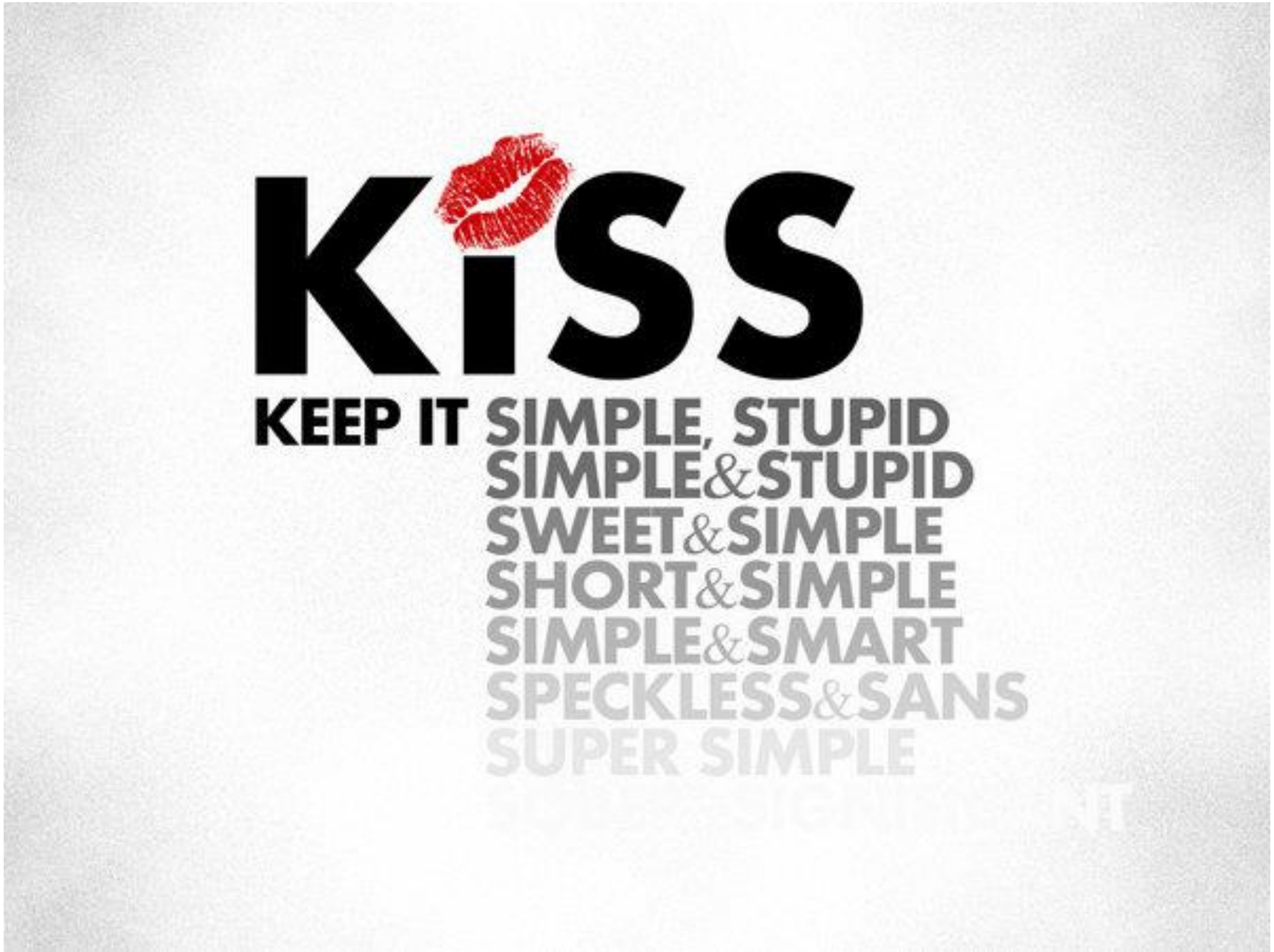
JAVA LAMBDA ~~RULES~~ GUIDELINES


The code is more what you'd call
guidelines than actual rules.

LAMBDA is an elegant, specialized tool.

No documentation –
intended to be
self-explanatory, simple!




KISS
KEEP IT SIMPLE, STUPID
SIMPLE & STUPID
SWEET & SIMPLE
SHORT & SIMPLE
SIMPLE & SMART
SPECKLESS & SANS
SUPER SIMPLE

If an expression:

1) Is not self-explanatory

2) Is longer than a few lines

If an expression:

1) Is not self-explanatory

2) Is longer than a few lines

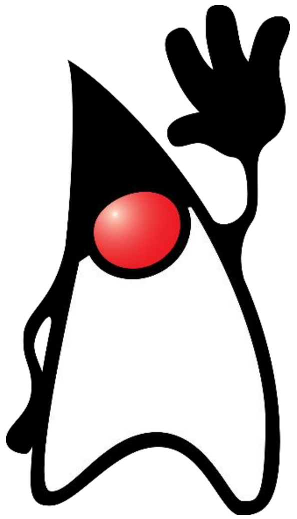
If an expression:

1) Is not self-explanatory

2) Is longer than a few lines

Three lines – reasonable maximum.

One line – ideal!

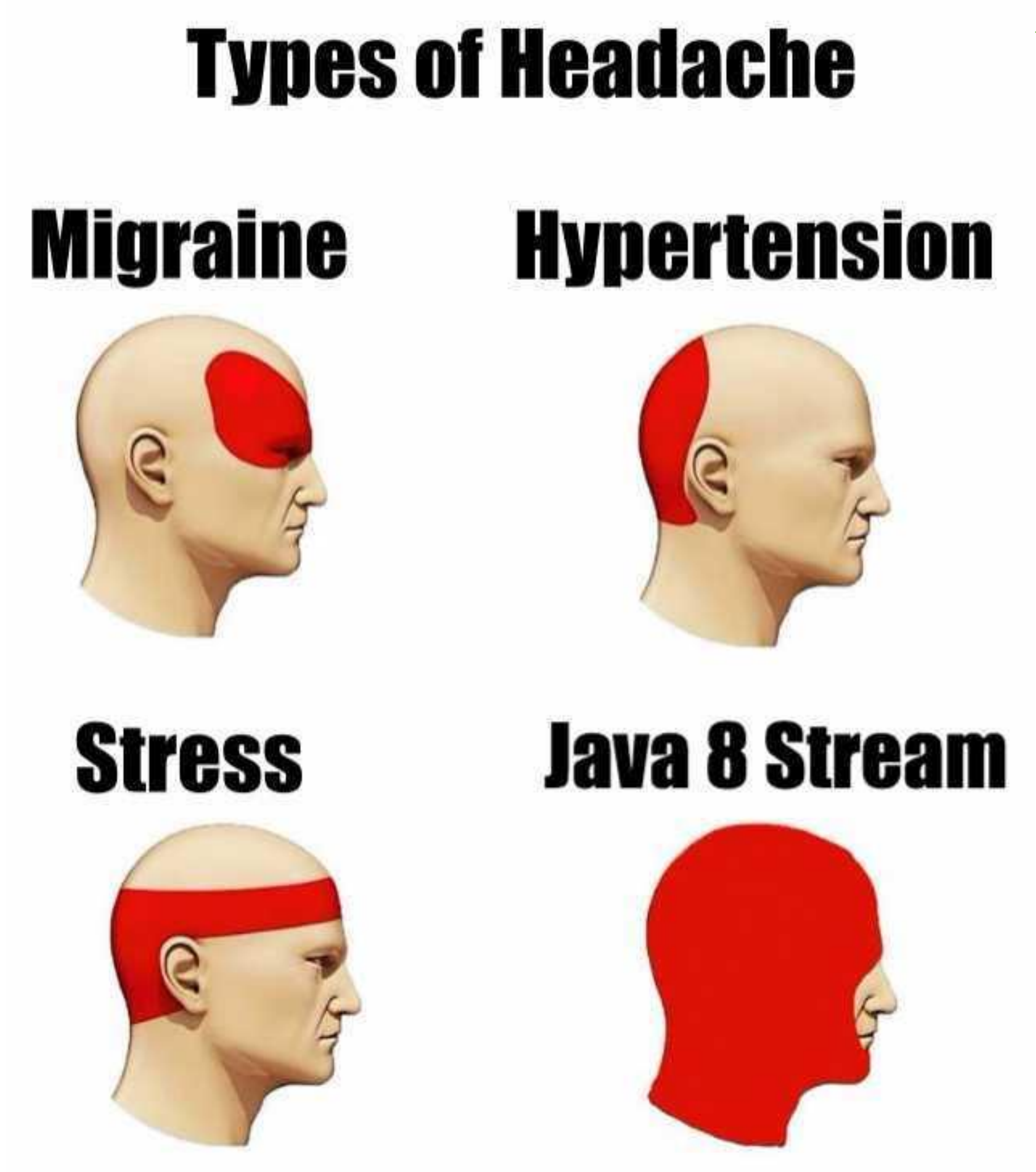


RECAP

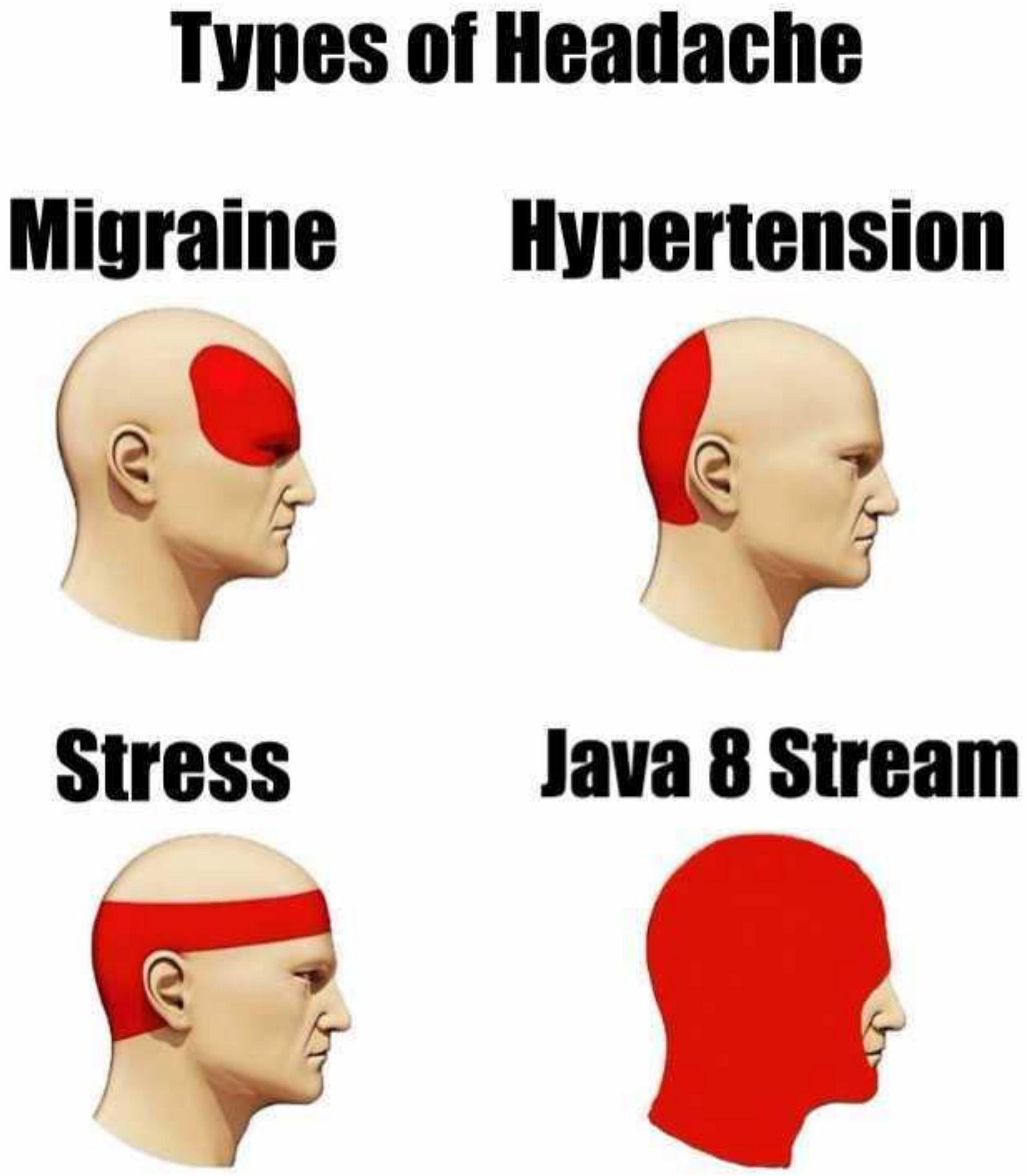
- Lambda expression is an *anonymous, parameterized* ***function***
- Keeps code clean, simple. Removes boilerplate.
- Can be used in place of *functional interfaces*
- *java.util.function* package contains standard implementations
- Can be further simplified via ::method references

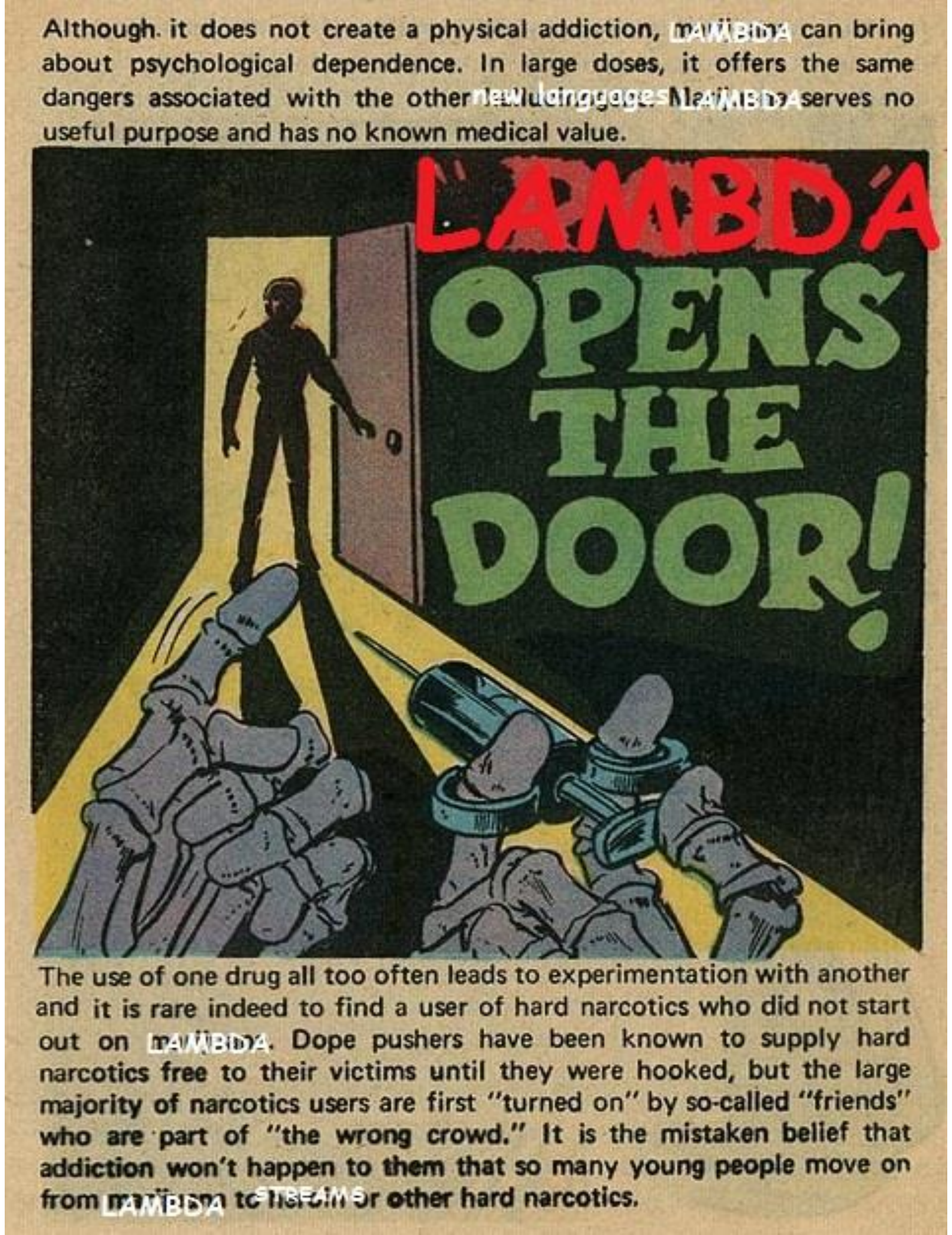
СБЕРБАНК ТЕХНОЛОГИИ

STREAM API



But there's a way…

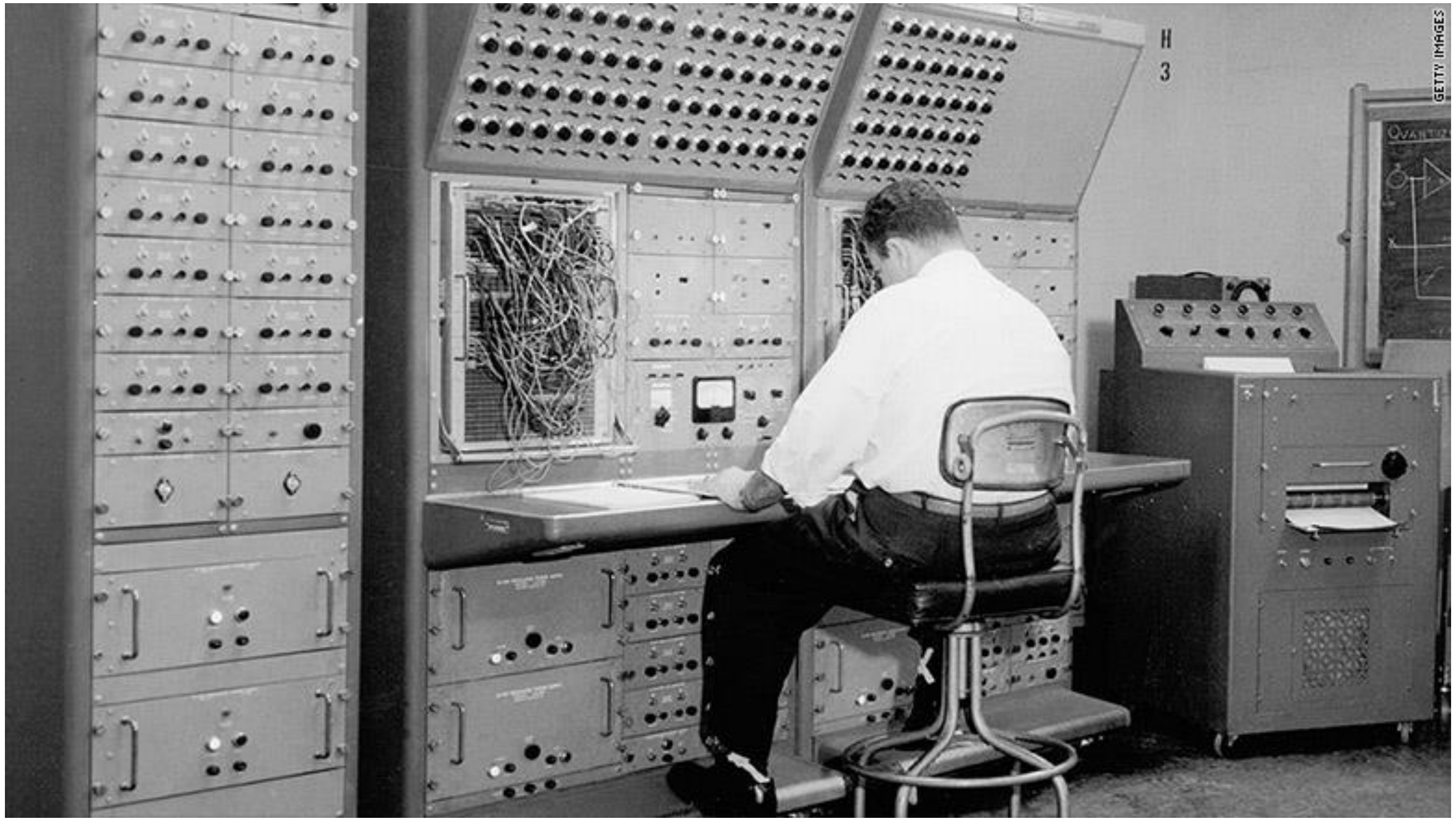




Although it does not create a physical addiction, LAMBDA can bring about psychological dependence. In large doses, it offers the same dangers associated with the other new languages. LAMBDA serves no useful purpose and has no known medical value.

The use of one drug all too often leads to experimentation with another and it is rare indeed to find a user of hard narcotics who did not start out on LAMBDA. Dope pushers have been known to supply hard narcotics free to their victims until they were hooked, but the large majority of narcotics users are first "turned on" by so-called "friends" who are part of "the wrong crowd." It is the mistaken belief that addiction won't happen to them that so many young people move on from LAMBDA to STREAMS or other hard narcotics.

WORKING WITH COLLECTIONS PRIOR TO JAVA 8



СБЕРБАНК ТЕХНОЛОГИИ

WORKING WITH COLLECTIONS AFTER JAVA 8

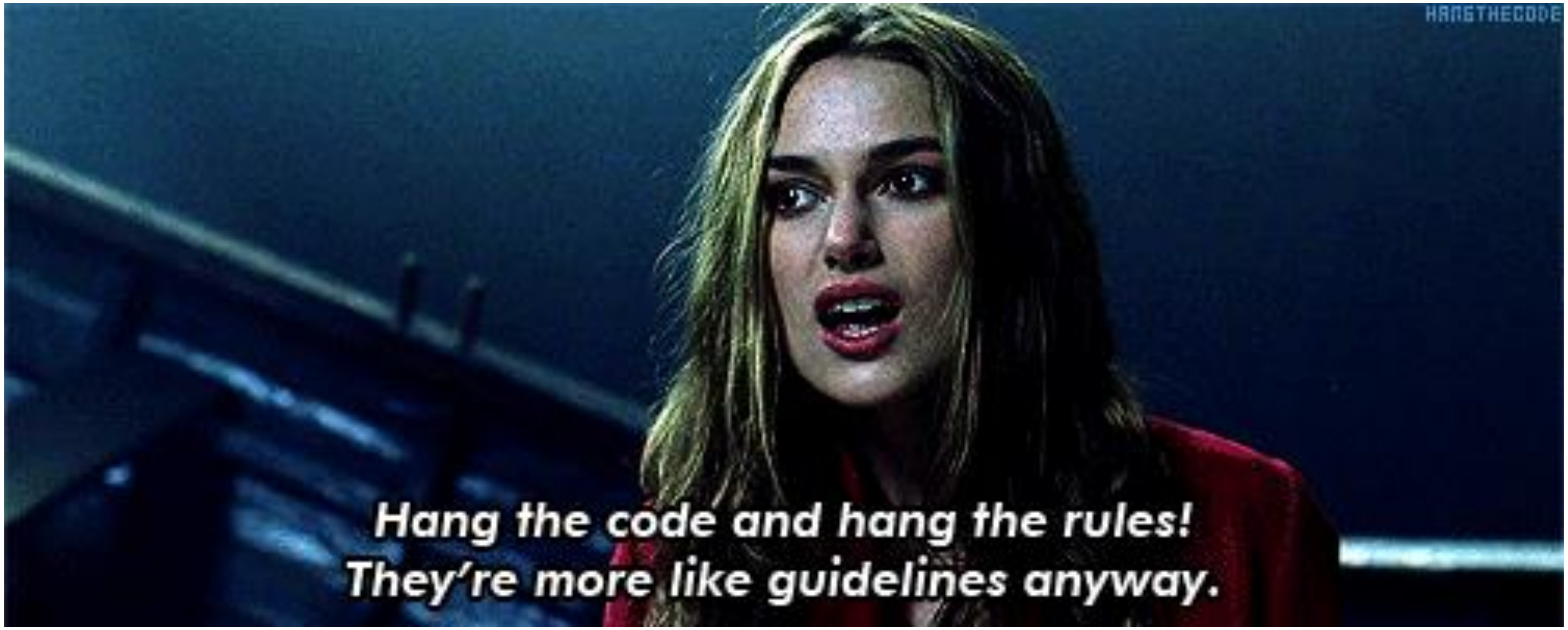
STREAM API

- Declarative programming
- Complex data pipelines
- Internal iteration

JAVA STREAM
RULES

СБЕРБАНК ТЕХНОЛОГИИ

JAVA STREAM ~~RULES~~ GUIDELINES


HANGTHECODE
Hang the code and hang the rules!
They're more like guidelines anyway.

MUTATION

СБЕРБАНК ТЕХНОЛОГИИ

MUTATION

СБЕРБАНК ТЕХНОЛОГИИ

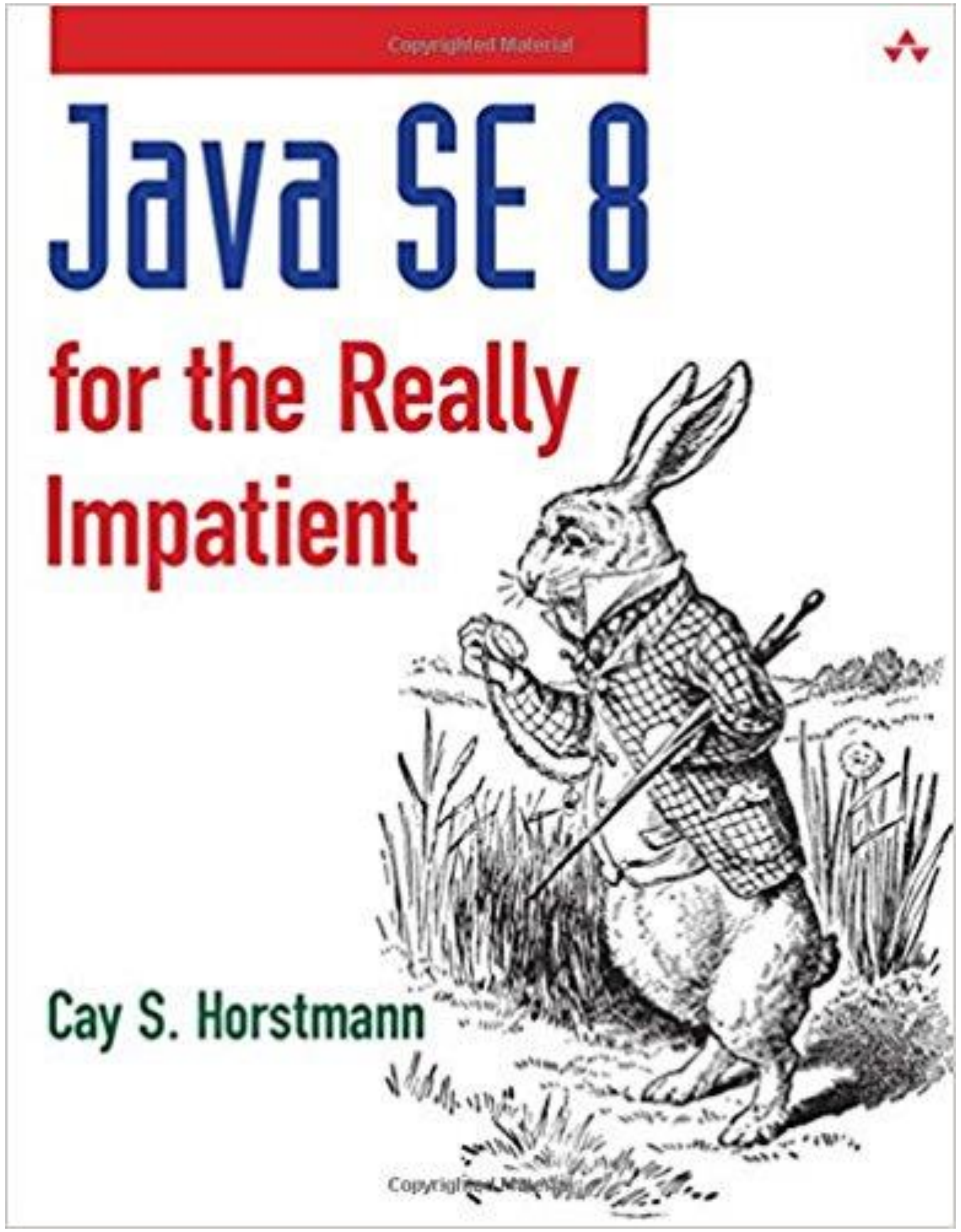
.stream()

.stream()

.parallelStream()

Learning Java 8

СБЕРБАНК ТЕХНОЛОГИИ



СБЕРБАНК ТЕХНОЛОГИИ

Venkat Subramaniam

agiledeveloper.com