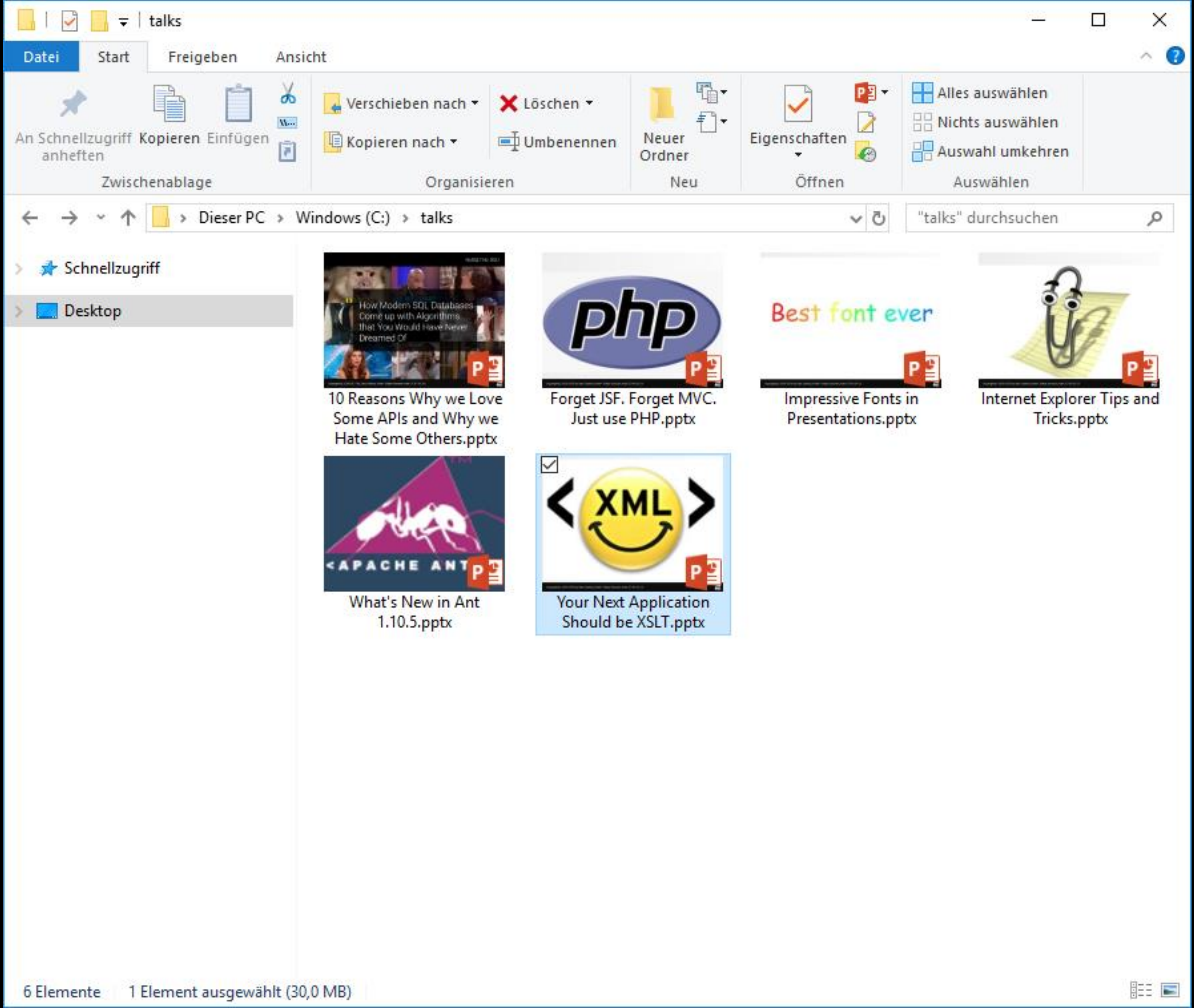
Why Your Next Application Should be Written With XSLT



10 Reasons Why we Love Some APIs and Why we Hate Some Others.pptx

Forget JSF. Forget MVC. Just use PHP.pptx

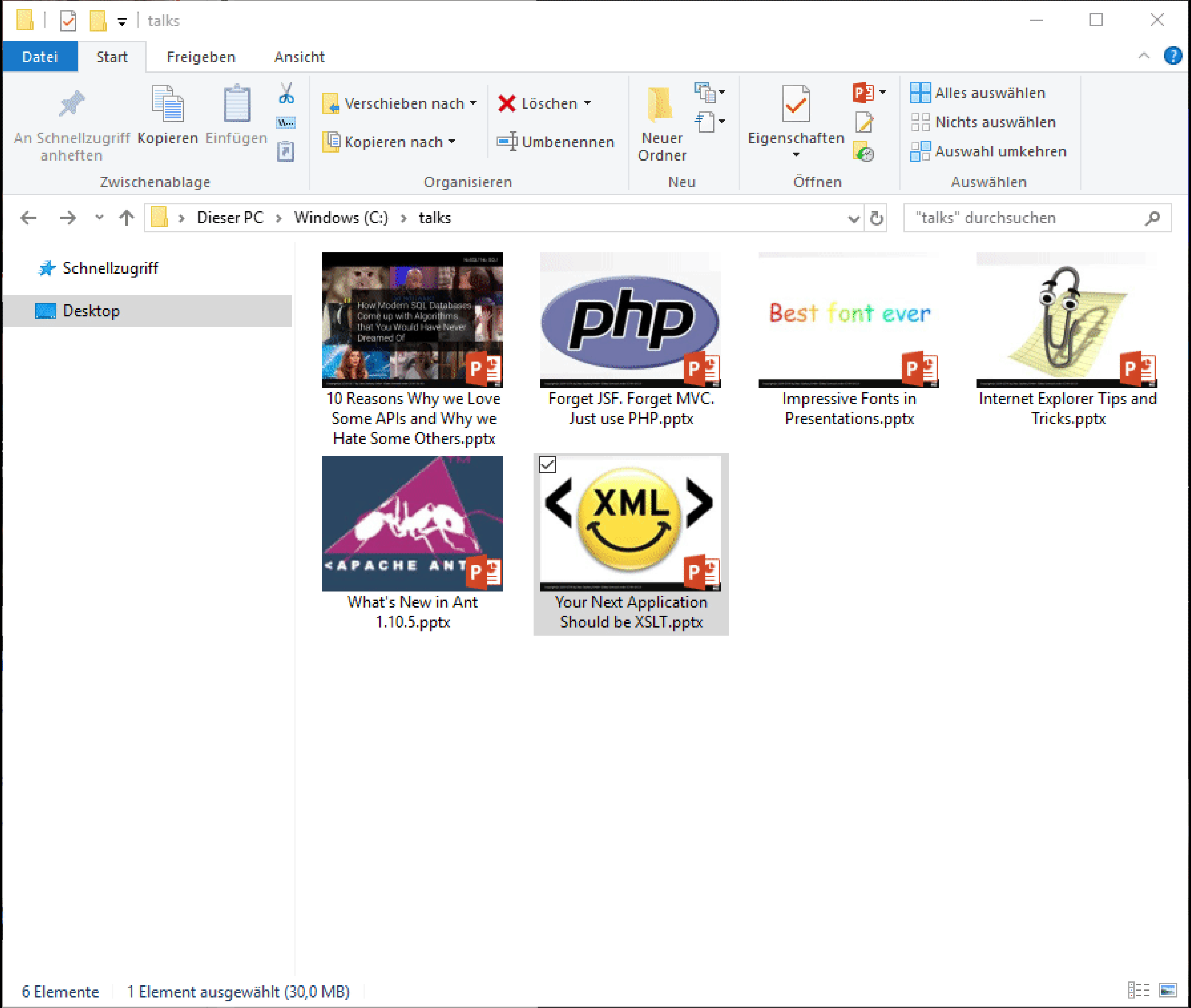
Impressive Fonts in Presentations.pptx

Internet Explorer Tips and Tricks.pptx

What's New in Ant 1.10.5.pptx

☑ Your Next Application Should be XSLT.pptx

6 Elemente   1 Element ausgewählt (30,0 MB)



10 Reasons Why we Love Some APIs and Why we Hate Some Others.pptx

Forget JSF. Forget MVC. Just use PHP.pptx

Impressive Fonts in Presentations.pptx

Internet Explorer Tips and Tricks.pptx

What's New in Ant 1.10.5.pptx

☑ Your Next Application Should be XSLT.pptx

6 Elemente 1 Element ausgewählt (30,0 MB)

10 Reasons Why we Love Some APIs and Why we Hate Some Others

Jokes

You know…

I rehearsed these jokes with my wife

Jokes

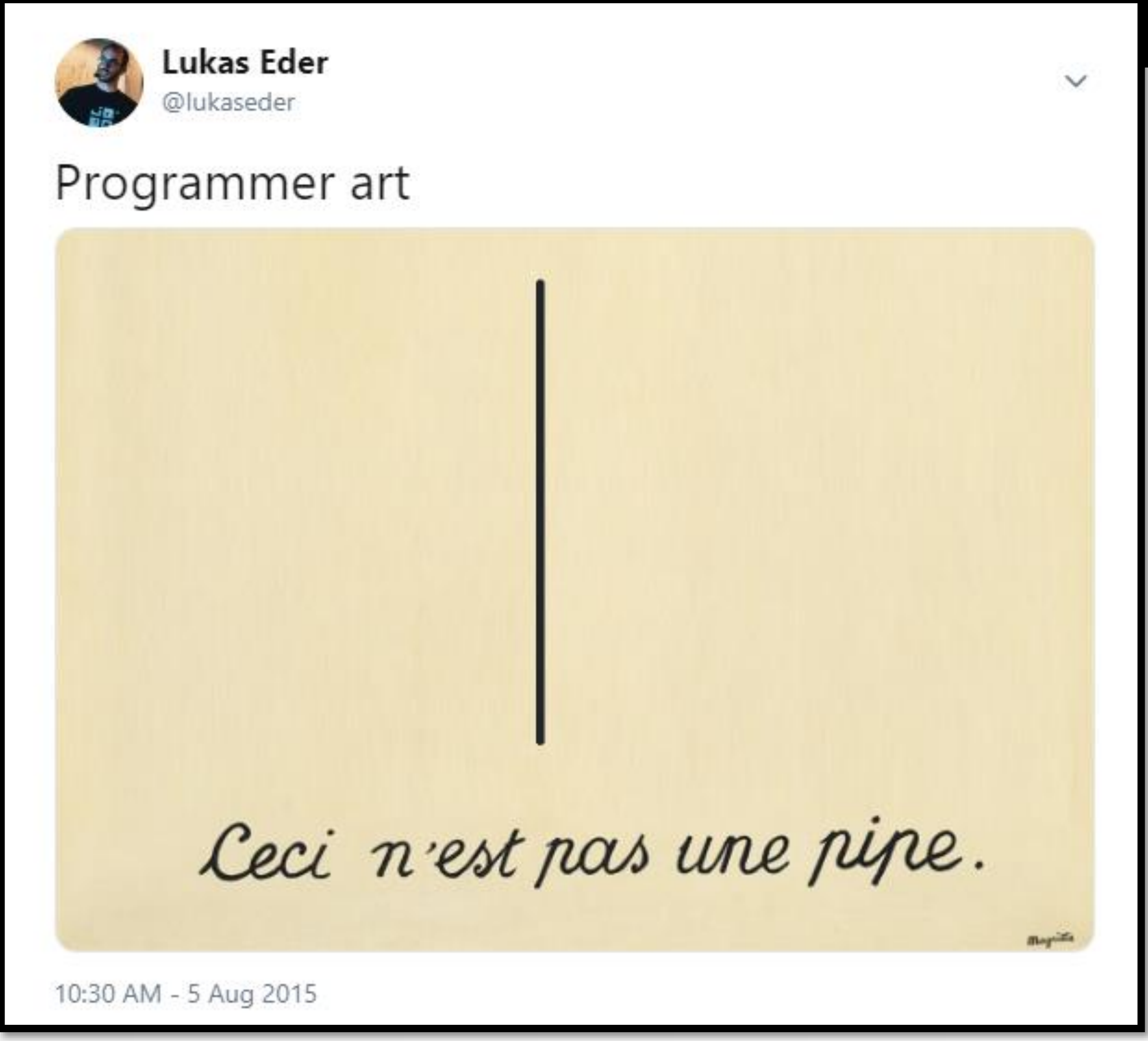
Lukas Eder
@lukaseder

Live footage of my wife, when I try to explain my programming jokes to her

2:54 PM - 15 Mar 2019

6 Retweets 32 Likes

Jokes



Lukas Eder
@lukaseder

Programmer art

10:30 AM - 5 Aug 2015

Good or bad?

```
var dir = new File(".");
for (var file : dir.list()) {
  System.out.println(file);
}
```

Good or bad?

```
var dir = new File("C:/tmp");
for (var file : dir.list()) {
  System.out.println(file);
}
```

Good or bad?

```
var dir = new File("C:/tmp");
for (var file : dir.list()) {
  System.out.println(file);
}
```

NullPointerException = File does not exist

Good or bad?

Terrible idea:

Returning a null array

Good or bad?

ok ok ok

This was old JDK API.
We're doing better now.

Good or bad?

```
Stream.of(1, 2, 3)
      .skip(1)
      .forEach(System.out::println);
```

Good or bad?

```
Stream.of(1, 2, 3)
      .skipUntil(t -> t == 2)
      .forEach(System.out::println);
```

Good or bad?

```
Stream.of(1, 2, 3)
//    .skipUntil(t -> t == 2) Nope!
      .forEach(System.out::println);
```

Good or bad?

```
Stream.of(1, 2, 3)
//    .skipUntil(t -> t == 2) Nope!
      .skipWhile(t -> t < 2)
      .forEach(System.out::println);
```

Good or bad?

```java
Stream.of(1, 2, 3)
//    .skipUntil(t -> t == 2) Nope!
//    .skipWhile(t -> t < 2)  Nope!
      .forEach(System.out::println);
```

Good or bad?

```
Stream.of(1, 2, 3)
//    .skipUntil(t -> t == 2) Nope!
//    .skipWhile(t -> t < 2)  Nope!
      .dropWhile(t -> t < 2)
      .forEach(System.out::println);
```

Good or bad?

Terrible idea:

Inconsistent naming

Good or bad?

```
// All input elements
$("input");
```

Good or bad?

```
// All input elements
$("input");

// Where they’re contained
$("input").parent();
```

Good or bad?

```
// All input elements
$("input");

// Where they're contained
$("input").parent();

// Only if they have names
$("input").filter("[name]");
```

Good or bad?

```
// All input elements
$("input");

// Where they’re contained
$("input").parent();

// Only if they have names
$("input").filter("[name]");

// Alternative
$("input[name]");
```

Good or bad?

```
// All input elements
$("input");

// Where they're contained
$("input").parent();

// Only if they have names
$("input").filter("[name]");

// Alternative
$("input[name]");

// Mapping / extracting information from contents
$("input[name]").map((i, e) => e.name);
```

Good or bad?

This feels great!

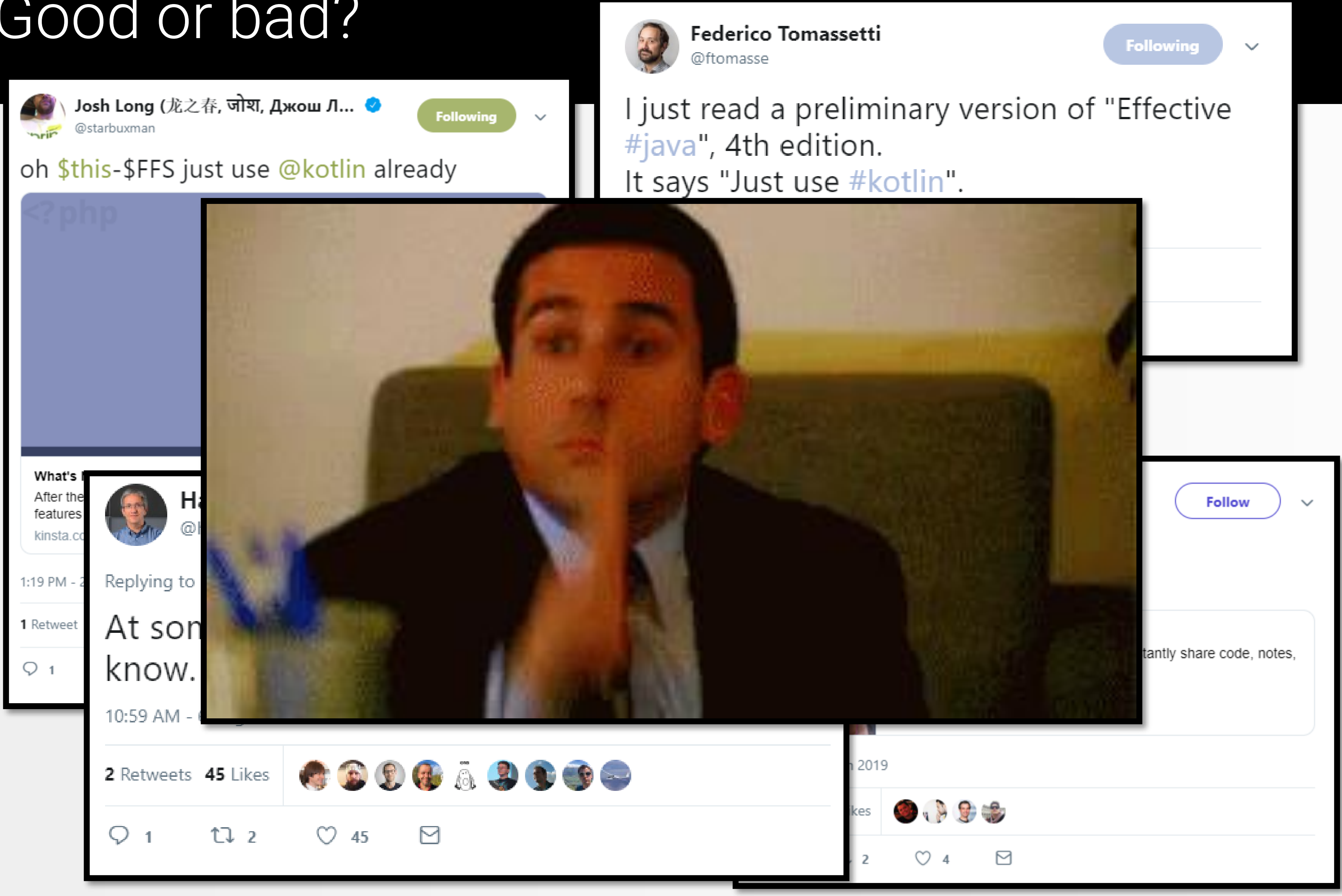
Good or bad?

But why?

Good or bad?

Not just about APIs.
About languages too.

Good or bad?



Josh Long (龙之春, जोश, Джош Л... @starbuxman

oh $this-$FFS just use @kotlin already

What's

After the

features

kinsta.c

1:19 PM - 2

1 Retweet

Federico Tomassetti @ftomasse

I just read a preliminary version of "Effective #java", 4th edition.
It says "Just use #kotlin".

H

@

Replying to

At son

know.

10:59 AM -

2 Retweets 45 Likes

tantly share code, notes,

2019

kes

Good or bad?

But today is about APIs.

User Experience

APIs are a crucial part of the UX (User Experience)

User Experience

> Programs must be written for people to read, and only incidentally for machines to execute

—Abelson & Sussman, "Structure and Interpretation of Computer Programs"

User Experience

> “Programs APIs must be written for people to read, and only incidentally for machines to execute”
>
> —me, just now

User Experience

> User experience (UX) refers to a person's emotions and attitudes about using a particular product, system or service

https://en.wikipedia.org/wiki/User_experience

User Experience

> User experience (UX) refers to a developer's emotions and attitudes about using a particular product, system or service

https://en.wikipedia.org/wiki/User_experience

User Experience

We developers do have emotions

User Experience

(or mostly attitudes)

User Experience

Don’t let them tell you otherwise

https://en.wikipedia.org/wiki/User_experience

User Experience

In fact,

An anecdote about developer attitudes

User Experience

A Kotlin Programmer

User Experience

A Kotlin Programmer, an IntelliJ user,

User Experience

A Kotlin Programmer, an IntelliJ user, and a Mac User

User Experience

A Kotlin Programmer, an IntelliJ user, and a Mac User went into a bar.

User Experience

A Kotlin Programmer, an IntelliJ user, and a Mac User went into a bar.

How do I know?

User Experience

A Kotlin Programmer, an IntelliJ user, and a Mac User went into a bar.

How do I know?

After 1 Minute

User Experience

A Kotlin Programmer, an IntelliJ user, and a Mac User went into a bar.

How do I know?

After 1 Minute, the whole f**king bar knew.

User Experience

> "User experience" encompasses all aspects of the end-user's interaction with the company, its services, and its products.

https://www.nngroup.com/articles/definition-user-experience/

User Experience

> "User experience" encompasses all aspects of the ~~end-user's~~ programmer's interaction with the company, its services, and its products.

https://www.nngroup.com/articles/definition-user-experience/

Usability

> Usability is defined by **5 quality components**:

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Usability

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Usability

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Usability

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?
>
> Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Usability

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?
>
> Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
>
> Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Usability

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?
>
> Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
>
> Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
>
> Satisfaction: How pleasant is it to use the design?

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Prejudices

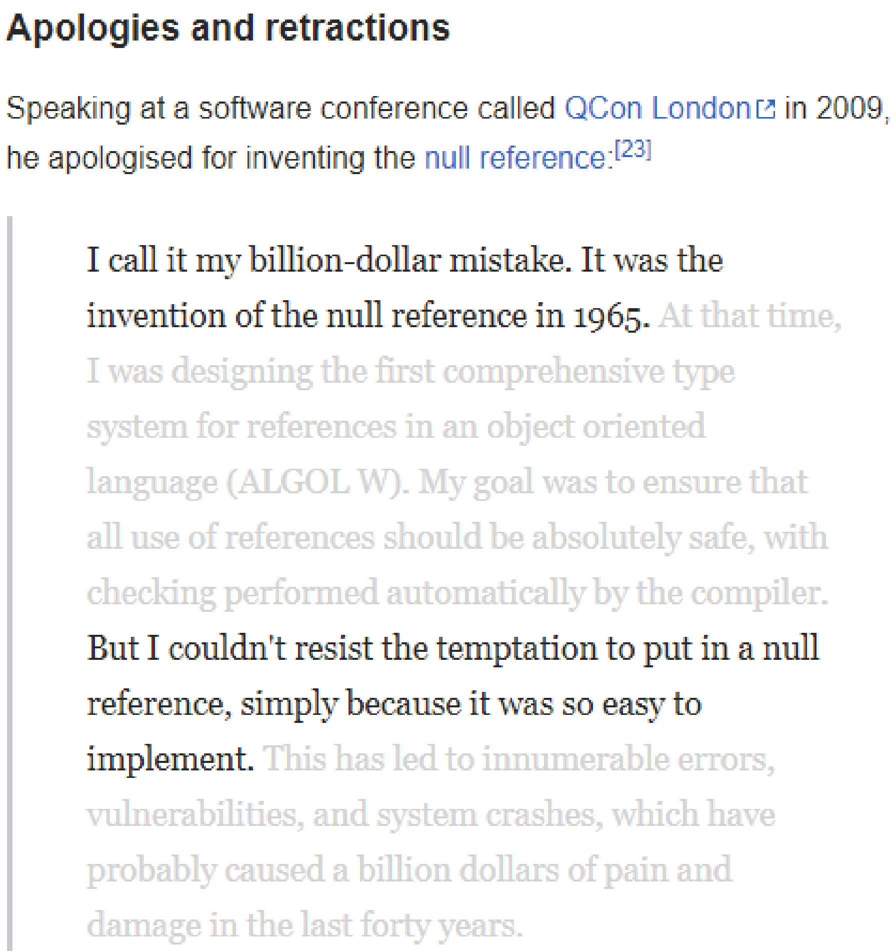
An important observation first

Prejudices

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?
>
> Memorability: When ers return to the design after a period of not usin ow easily can they reestablish proficiency?
>
> Errors: How rs do users make, how severe are these err easily can they recover from the errors?

These things aren't about technical details

Prejudices – null

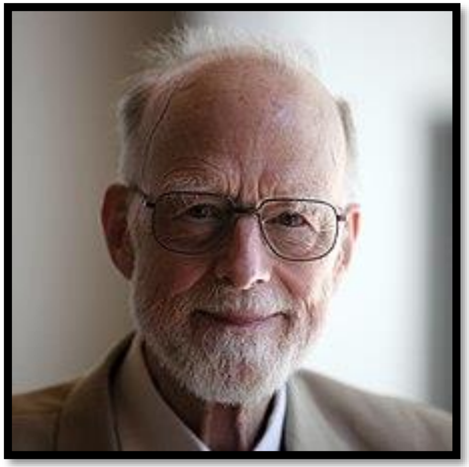


Apologies and retractions

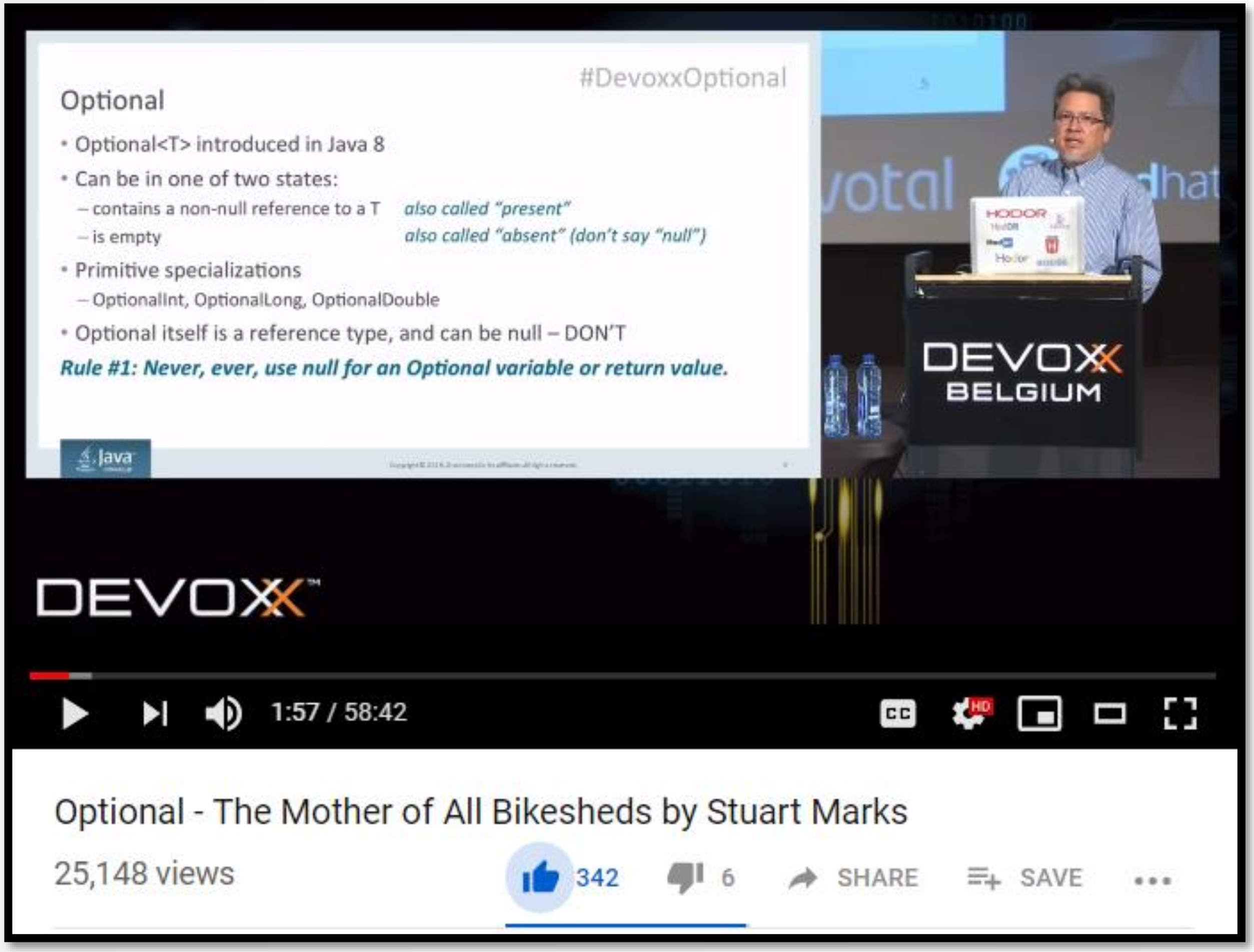
Speaking at a software conference called QCon London in 2009, he apologised for inventing the null reference:[23]

> I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

https://en.wikipedia.org/wiki/Tony_Hoare



Prejudices – null



#DevoxxOptional

Optional

- Optional<T> introduced in Java 8
- Can be in one of two states:
  - contains a non-null reference to a T — *also called "present"*
  - is empty — *also called "absent" (don't say "null")*
- Primitive specializations
  - OptionalInt, OptionalLong, OptionalDouble
- Optional itself is a reference type, and can be null – DON'T

Rule #1: Never, ever, use null for an Optional variable or return value.

Optional - The Mother of All Bikesheds by Stuart Marks

25,148 views

https://www.youtube.com/watch?v=Ej0sss6cq14

Prejudices – null

Null is not bad per se

Prejudices – null

Null is bad when it is unexpected

… You still want some special «absent» value

Prejudices

Let’s not bikeshed

Picture by Jeremy Segrot (CC BY 2.0) https://www.flickr.com/photos/126337928@N05/42791980705

Outline

10 Reasons Why we Love Some APIs and Why we Hate Some Others

1. Naming
2. Simplicity
3. Do One Thing
4. Types
5. Discoverability
6. Error Handling
7. Consistency
8. Convenience
9. Compatibility
10. Documentation

Outline

10 Reasons Why we Love Some APIs and Why we Hate Some Others

1. Naming
2. Simplicity
3. Do One Thing
4. Types
5. Discoverability
6. Error Handling
7. Consistency
8. Convenience
9. Compatibility
10. Documentation

Colloquially NSDTDECCCD

D is silent

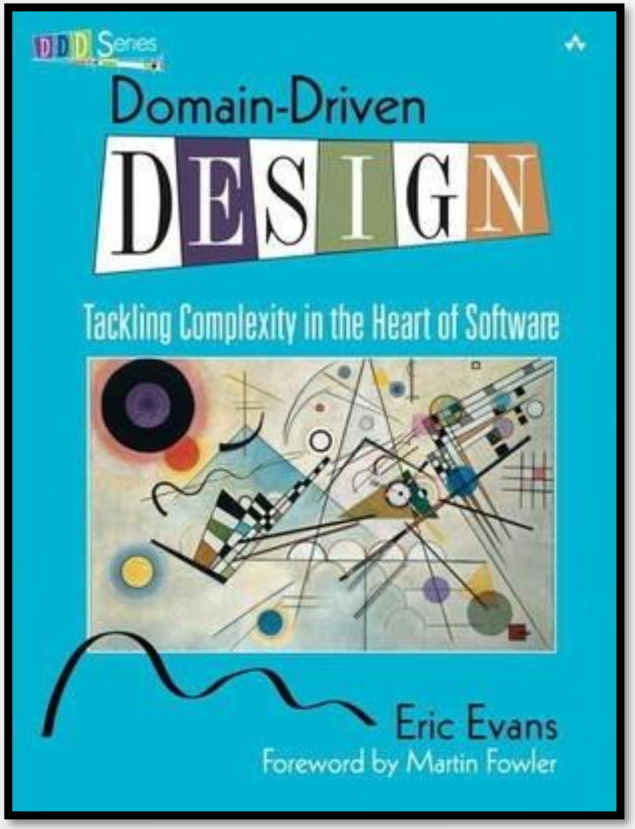
1. Naming

This is actually really simple.

… and simplicity is hard as we’ll soon see!

1. Naming

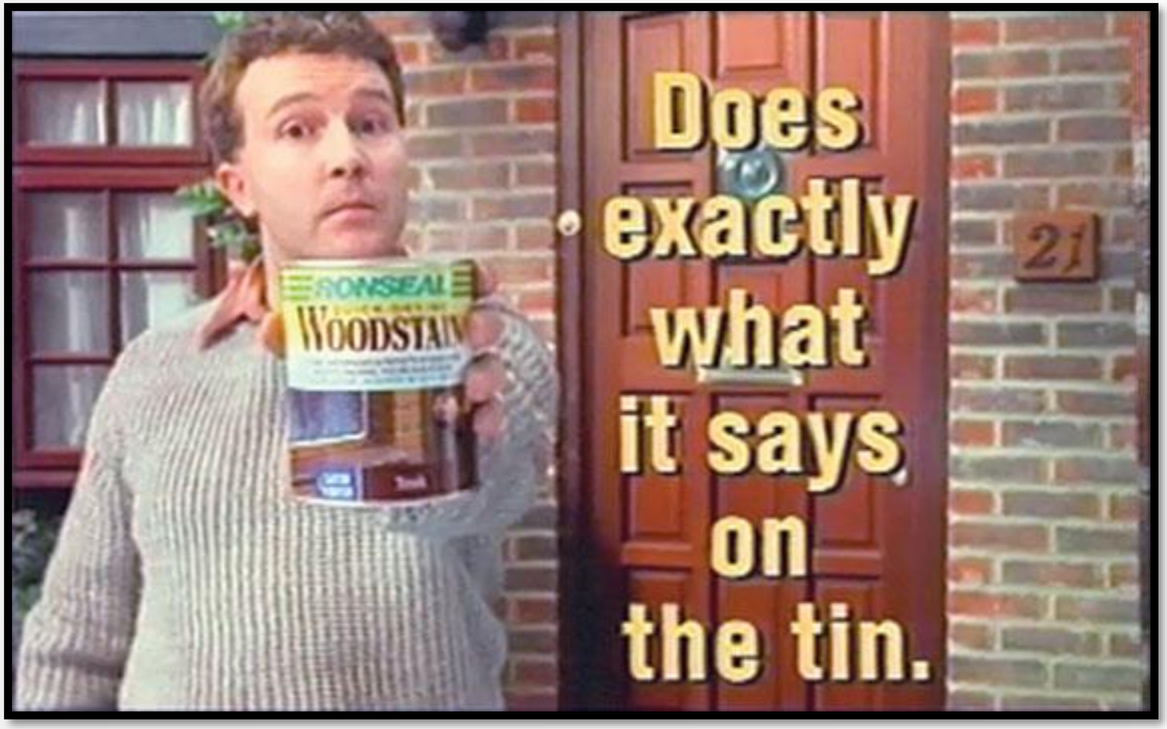
1. Naming – Ubiquitous Language


DDD Series
Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

> “Ubiquitous Language
>
> A language structured around the domain model and used by all team members to connect all the activities of the team with the software.”

https://en.wikipedia.org/wiki/Domain-driven_design

1. Naming – Why is it important?



1. Naming – Don’t do this

```
public interface Stream<T> {

  // Looks like Informix :-)
  // SELECT SKIP 10 LIMIT 10 * FROM my_table
  Stream<T> skip(long n);
  Stream<T> limit(long maxSize);





}
```

1. Naming – Don’t do this

```
public interface Stream<T> {

  // Looks like Informix :-)
  // SELECT SKIP 10 LIMIT 10 * FROM my_table
  Stream<T> skip(long n);
  Stream<T> limit(long maxSize);

  // These have been added in Java 9
  // They follow Scala naming conventions
  default Stream<T> dropWhile(Predicate<? super T> predicate) {}
  default Stream<T> takeWhile(Predicate<? super T> predicate) {}
  ..
}
```

1. Naming

You know how this happens?

1. Naming

Every single time...

1. Naming – Consistency

When someone else touches my API…

I Am Devloper
@iamdevloper

when you visit a site you handed over 6 months ago and the client made some changes themselves

3:38 PM - 6 Jan 2019

2,162 Retweets 8,852 Likes

1. Naming – Can be fixed

It seems that the «language» has changed.

1. Naming – Can be fixed

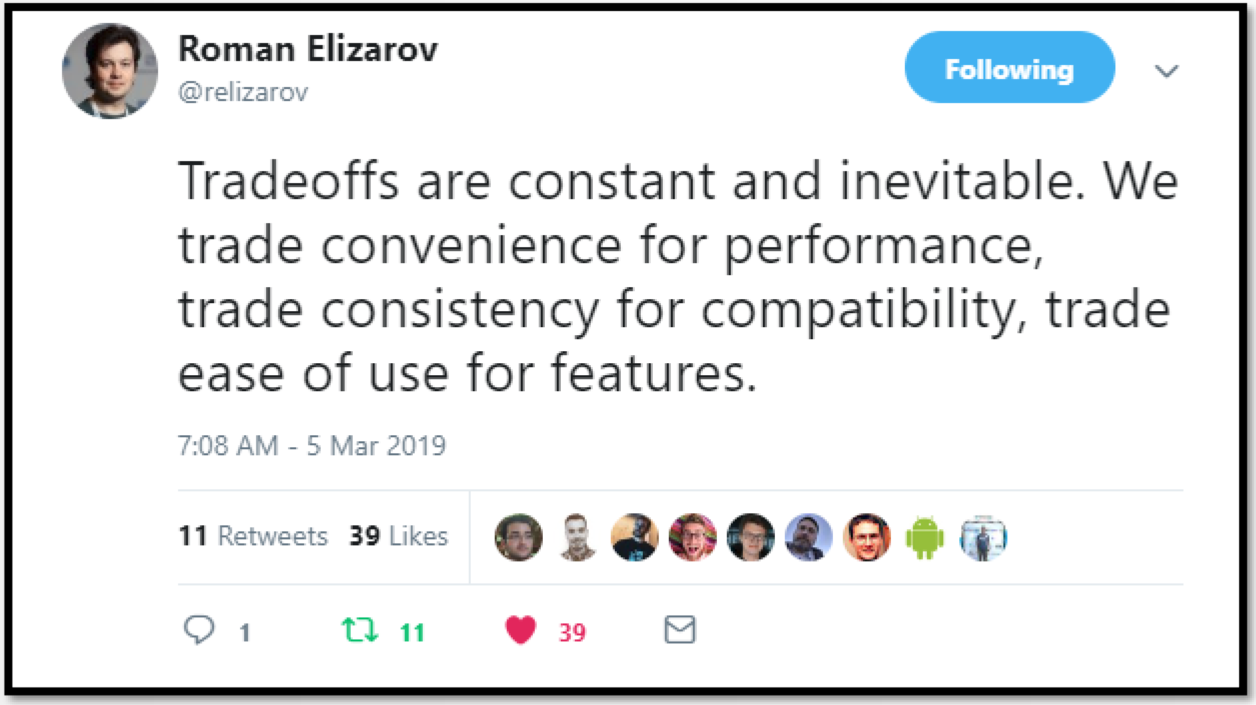
This can be fixed.

… but keep backwards compatibility in mind

1. Naming – Tradeoffs

In an API, sometimes, backwards compatibility is more important than good, consistent names

1. Naming – Tradeoffs



Roman Elizarov
@relizarov

Tradeoffs are constant and inevitable. We trade convenience for performance, trade consistency for compatibility, trade ease of use for features.

7:08 AM - 5 Mar 2019

11 Retweets 39 Likes

1. Naming – Consistency

Better get it right the first time.

And talk to each other!

1. Naming – Consistency

```
// Some monadic JDK types
```

1. Naming – Consistency

```
// Some monadic JDK types
public interface Stream<T> {
  <R> Stream<R> map(Function<? super T, ? extends R> mapper);
  <R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>> mapper);
}
```

1. Naming – Consistency

```
// Some monadic JDK types
public interface Stream<T> {
  <R> Stream<R> map(Function<? super T, ? extends R> mapper);
  <R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>> mapper);
}

public final class Optional<T> {
  public <U> Optional<U> map(Function<? super T, ? extends U> mapper);
  public <U> Optional<U> flatMap(Function<? super T, ? extends Optional<? extends U>> mapper);
}
```

1. Naming – Consistency

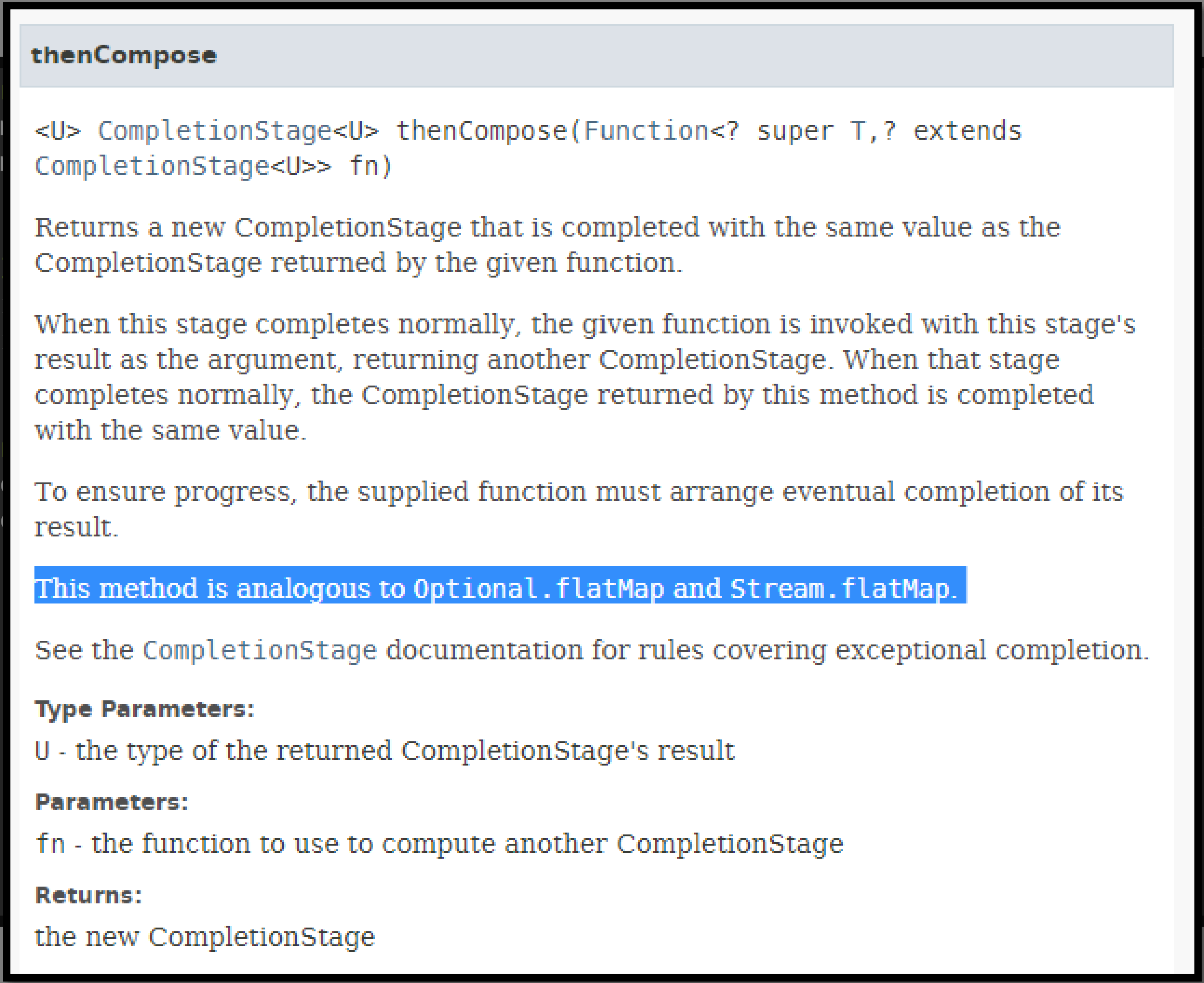
```
// Some monadic JDK types
public interface Stream<T> {
  <R> Stream<R> map(Function<? super T, ? extends R> mapper);
  <R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>> mapper);
}

public final class Optional<T> {
  public <U> Optional<U> map(Function<? super T, ? extends U> mapper);
  public <U> Optional<U> flatMap(Function<? super T, ? extends Optional<? extends U>> mapper);
}

public interface CompletionStage<T> {
  <U> CompletionStage<U> ...      (Function<? super T, ? extends U> fn);
  <U> CompletionStage<U> ...        (Function<? super T, ? extends CompletionStage<U>> fn);
}
```

1. Naming – Consistency

```java
// Some monadic JDK types
public interface Stream<T> {
  <R> Stream<R> map(Function<? super T, ? extends R> mapper);
  <R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>> mapper);
}

public final class Optional<T> {
  public <U> Optional<U> map(Function<? super T, ? extends U> mapper);
  public <U> Optional<U> flatMap(Function<? super T, ? extends Optional<? extends U>> mapper);
}

public interface CompletionStage<T> {
  <U> CompletionStage<U> thenApply(Function<? super T, ? extends U> fn);
  <U> CompletionStage<U> thenCompose(Function<? super T, ? extends CompletionStage<U>> fn);
}
```

1. Naming – Consistency

```
public inte
  <R> Strea
  <R> Strea
}

public fina
  public <U
  public <U                                                        > mapper);
}

public inte
  <U> Compl
  <U> Compl                                                        >> fn);
}
```



thenCompose

```
<U> CompletionStage<U> thenCompose(Function<? super T,? extends
CompletionStage<U>> fn)
```

Returns a new CompletionStage that is completed with the same value as the CompletionStage returned by the given function.

When this stage completes normally, the given function is invoked with this stage's result as the argument, returning another CompletionStage. When that stage completes normally, the CompletionStage returned by this method is completed with the same value.

To ensure progress, the supplied function must arrange eventual completion of its result.

This method is analogous to `Optional.flatMap` and `Stream.flatMap`.

See the `CompletionStage` documentation for rules covering exceptional completion.

Type Parameters:

`U` - the type of the returned CompletionStage's result

Parameters:

`fn` - the function to use to compute another CompletionStage

Returns:

the new CompletionStage

1. Naming – Examples from jOOQ

```
// What's jOOQ? It's this internal DSL to create type safe, dynamic,
// vendor agnostic SQL statements in Java

ctx.select(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME, count())
   .from(AUTHOR)
   .join(BOOK).on(AUTHOR.ID.eq(BOOK.AUTHOR_ID))
   .where(BOOK.LANGUAGE.eq("DE"))
   .and(BOOK.PUBLISHED.gt(date("2008-01-01")))
   .groupBy(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
   .having(count().gt(5))
   .orderBy(AUTHOR.LAST_NAME.asc().nullsFirst())
   .limit(2)
   .offset(1)
   .fetch();
```

1. Naming – Exa

Very obvious method names

```
// What’s jOOQ? I   nis internal DSL to create type safe, dynamic,
// vendor agn   ic SQL statements in Java

ctx.select(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME, count())
   .from(AUTHOR)
   .join(BOOK).on(AUTHOR.ID.eq(BOOK.AUTHOR_ID))
   .where(BOOK.LANGUAGE.eq("DE"))
   .and(BOOK.PUBLISHED.gt(date("2008-01-01")))
   .groupBy(AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
   .having(count().gt(5))
   .orderBy(AUTHOR.LAST_NAME.asc().nullsFirst())
   .limit(2)
   .offset(1)
   .fetch();
```

1. Naming – Wonderful API

1. Naming – Examples from jOOQ

Yet

1. Naming – Examples from jOOQ

```
// Field vs Column
public interface Field<T> { ... }
public class DSL {
  public static Field<String> substring(Field<String> str, ...) { ... }
}
```

1. Naming – Examples from jOOQ

```
// Field vs Column
public interface Field<T> { ... }
public class DSL {
  public static Field<String> substring(Field<String> str, ...) { ... }
}

// But then
ctx.createTable("my_table")
   .column("my_column", VARCHAR(10))
   .execute();
```

What now. Field or Column?

1. Naming – Interpretation

Bad naming in their API

Bad naming in my API

1. Naming – Examples from jOOQ

```
// Condition vs Predicate
// Currently:
public interface Condition { ... }
```

1. Naming – Examples from jOOQ

```
// Condition vs Predicate
// Currently:
public interface Condition { ... }

// “Better”:
public interface Predicate { ... }
```

1. Naming – Examples from jOOQ

```
// Condition vs Predicate
// Currently:
public interface Condition { ... }

// “Better”:
public interface Predicate { ... }
```

Me, every day,
with this idea

1. Naming – Why not?

Changing the API is easy.

1. Naming – Why not?

But you have to:

1. Naming – Why not?

But you have to:

- Change the docs (please use HTTP 301 Redirect and don’t create dead links!)

1. Naming – Why not?

But you have to:

- Change the docs (please use HTTP 301 Redirect and don’t create dead links!)
- Don’t forget «external doc» like Stack Overflow

1. Naming – Why not?

But you have to:

- Change the docs (please use HTTP 301 Redirect and don't create dead links!)
- Don't forget «external doc» like Stack Overflow
- Write good release notes / upgrade notes

1. Naming – Why not?

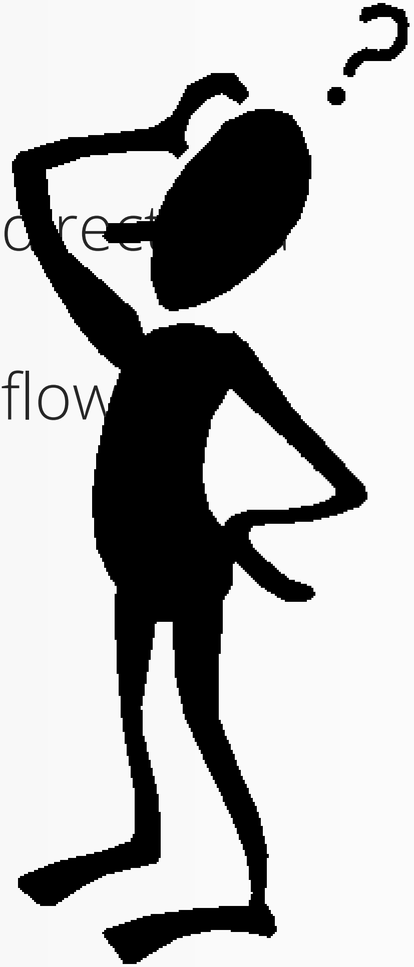
But you have to:

- Change the docs (please use HTTP 301 Redirect and don’t create dead links!)
- Don’t forget «external doc» like Stack Overflow
- Write good release notes / upgrade notes
- Keep the old name around for backwards compatibility

1. Naming – Why not?

But you have to:

- Change the docs (please use HTTP 301 Redirect [illegible] don’t create dead links!)
- Don’t forget «external doc» like Stack Overflow
- Write good release notes / upgrade notes
- Keep the old name around for backwards compatibility



Is it worth it?

1. Naming – Do this (JSR-310)

```
// “Does what it says it does on the tin,” consistently.

public final class LocalDate {
  public static LocalDate now() {}
  public static LocalDate parse(CharSequence text) {}
  public static LocalDate of(int year, int month, int dayOfMonth) {}
}
public final class LocalTime {
  public static LocalTime now() {}
  public static LocalTime parse(CharSequence text) {}
  public static LocalDate of(int hour, int minute) {}
}
public final class Instant {
  public static Instant now() {}
  public static Instant parse(CharSequence text) {}
  public static Instant ofEpochSecond(long epochSecond) {}
}
```

R-310)

«now» is very obvious

«parse» too

```
// “Does what it says it   s on

public final class LocalDate {
  public static LocalDate now() {}
  public static LocalDate parse(CharSequence text) {}
  public static LocalDate of(int year, int month, int dayOfMonth) {}
}
public final class LocalTi  {
  public static LocalTime      {}
  public static LocalTime      CharSequence text) {}
  public static LocalDate      our, int minute) {}
}
public final class Insta
  public stati
  public stati
  public stati
}
```

Create a value «of» its parts

1. Naming – Do this (JSR-310)

```
// Other “language”:
public final class LocalDate {
  public LocalDateTime atStartOfDay() {}
  public LocalDateTime atTime(int hour, int minute) {}

  public int get(TemporalField field) {}
  public int getDayOfMonth() {}
  public Month getMonth() {}

  public boolean isAfter(ChronoLocalDate other) {}
  public boolean isBefore(ChronoLocalDate other) {}

  public LocalDate minusDays(int daysToSubtract) {}
  public LocalDate minusMonths(int monthsToSubtract) {}

  public LocalDate withDayOfMonth(int dayOfMonth) {}
  public LocalDate withMonth(int month) {}
}
```

```
// Other “languag
public final class Lo    ate {
  public LocalDateTime a StartOfDay() {}
  public LocalDateTime atTime(int h    , int minute) {}

  public int get(TemporalField field) {
  public int getDayOfMonth() {}
  public Month getMonth() {}

  public boolean isAfter(ChronoLocalDate other) {}
  public boolean isBefore(ChronoLocalDa

  public LocalDate minusDays(int daysTo
  public LocalDate minusMonths(int mo

  public LocalDate withDayOfMonth(int dayOfMonth) {}
  public LocalDate withMonth(int
}
```

«at» creates a more precise value

«get» extracts a part

«is» runs a check

«minus» and «plus» arithmetic

«with» replaces a part

1. Naming – Do this (JSR-310)

«startOfDay»

«dayOfMonth»

«dayOfMonth»

```
// Other “language”:
public final class LocalDate {
  public LocalDateTime atStartOfDay() {}
  public LocalDateTime atTime(int hour, i

  public int get(TemporalField field) {}
  public int getDayOfMonth() {}
  public Month getMonth() {}

  public boolean isAfter(ChronoLocalDate other) {}
  public boolean isBefore(ChronoLocalDate other)

  public LocalDate minusDays(int daysToSubtract) {}
  public LocalDate minusMonths(int monthsToSubtract) {}

  public LocalDate wit      onth(int dayOfMonth) {}
  public LocalDate wit        month) {}
```

«day» != «dayOfMonth>

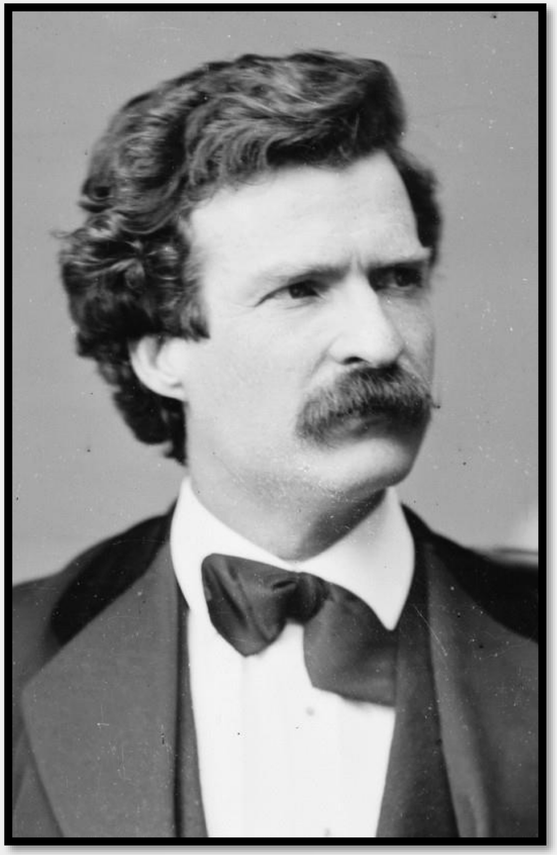
1. Naming

Better get it right from the beginning!

2. Simplicity

Simplicity

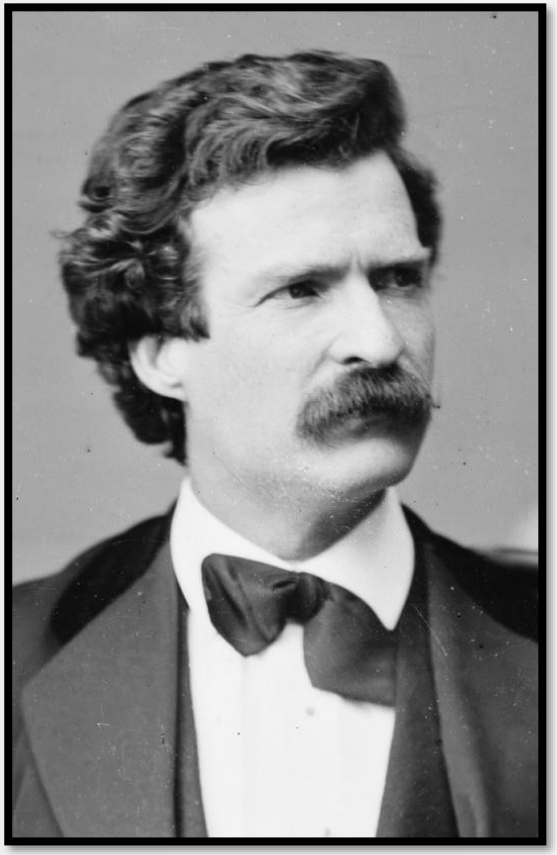
2. Simplicity – Technical View



> “I didn't have time to write a short letter, so I wrote a long one instead.”

—Mark Twain / https://en.wikipedia.org/wiki/Mark_Twain

2. Simplicity – Manager Version



> I didn't have time to think about who this is for, so I CC’ed everyone instead.

—Not Mark Twain / https://en.wikipedia.org/wiki/Mark_Twain

2. Simplicity – Spiritual View

> It is pointless to do with more what can be done with fewer.

—William of Ockham / https://en.wikiquote.org/wiki/William_of_Ockham

2. Simplicity – Spiritual View

> The more you have, the more you are occupied. The less you have, the more free you are

—Mother Teresa / https://en.wikipedia.org/wiki/Mother_Teresa

2. Simplicity

Why the spiritual context?

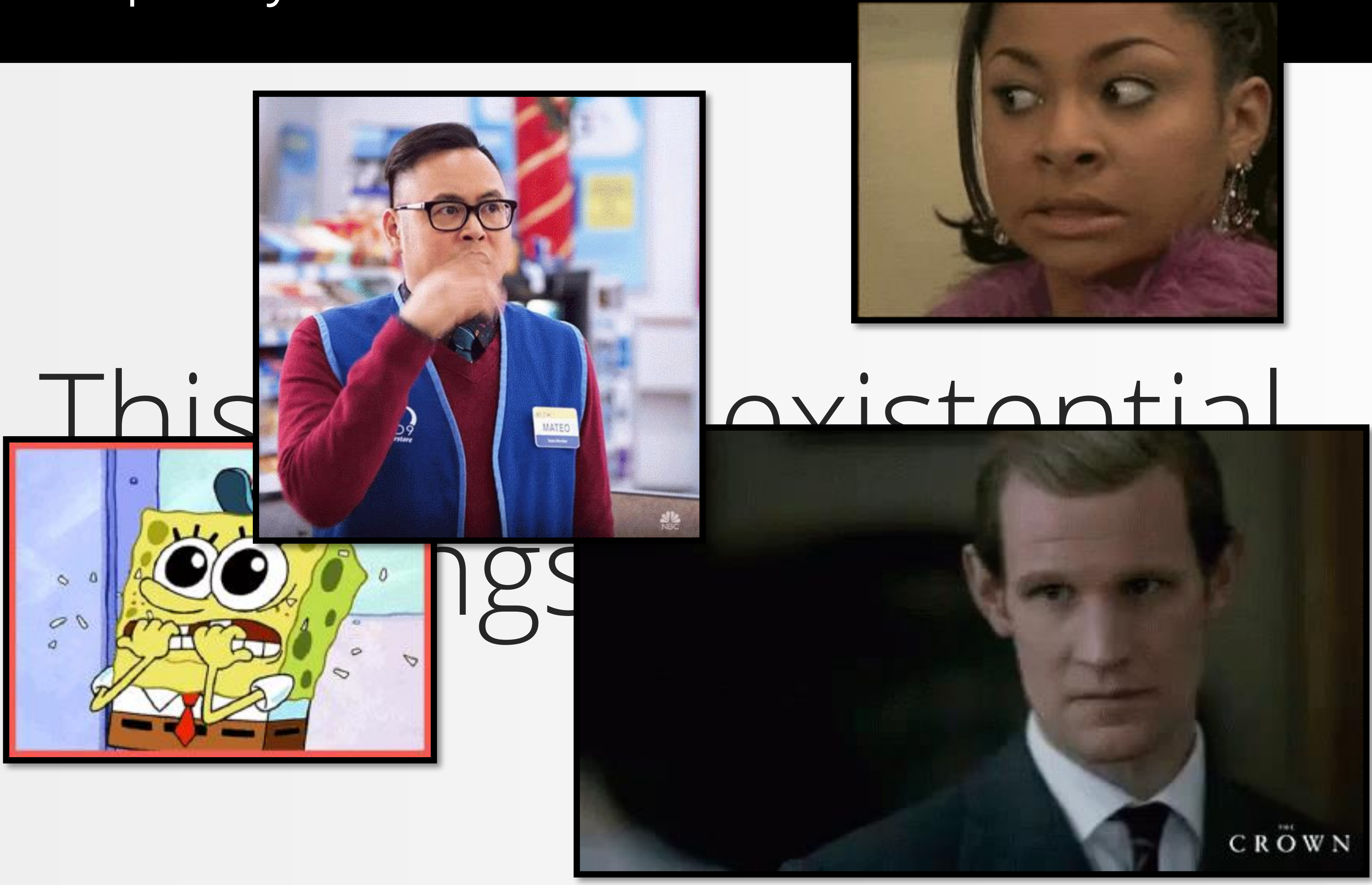
2. Simplicity – How not to be simple

```
public interface JavaCompiler
  extends Tool, OptionChecker {

  CompilationTask getTask(
    Writer out,
    JavaFileManager fileManager,
    DiagnosticListener<? super JavaFileObject> listener,
    Iterable<String> options,
    Iterable<String> classes,
    Iterable<? extends JavaFileObject> compilationUnits
  );

  ...
```

2. Simplicity

This causes existential angst in me

2. Simplicity



This existential

ngs

2. Simplicity – How not to be simple

```
public interface JavaCompiler
  extends Tool, OptionChecker {

  CompilationTask getTask(
    Writer out,
    JavaFileManager fileMan
    DiagnosticListener<? super          ileObject> listener,
    Iterable<String> options,
    Iterable<String> classes,
    Iterable<? extends JavaFileObject> compilationUnits
  );

  ...
```

6 Parameters

Stringly typed

Difficult to build types

2. Simplicity – Why is it important

None of these things are inherently bad.

2. Simplicity – Why is it important

The implementation works very well.

2. Simplicity – Why is it important

But we are humans

2. Simplicity – Why is it important

We can only keep so many things in our heads

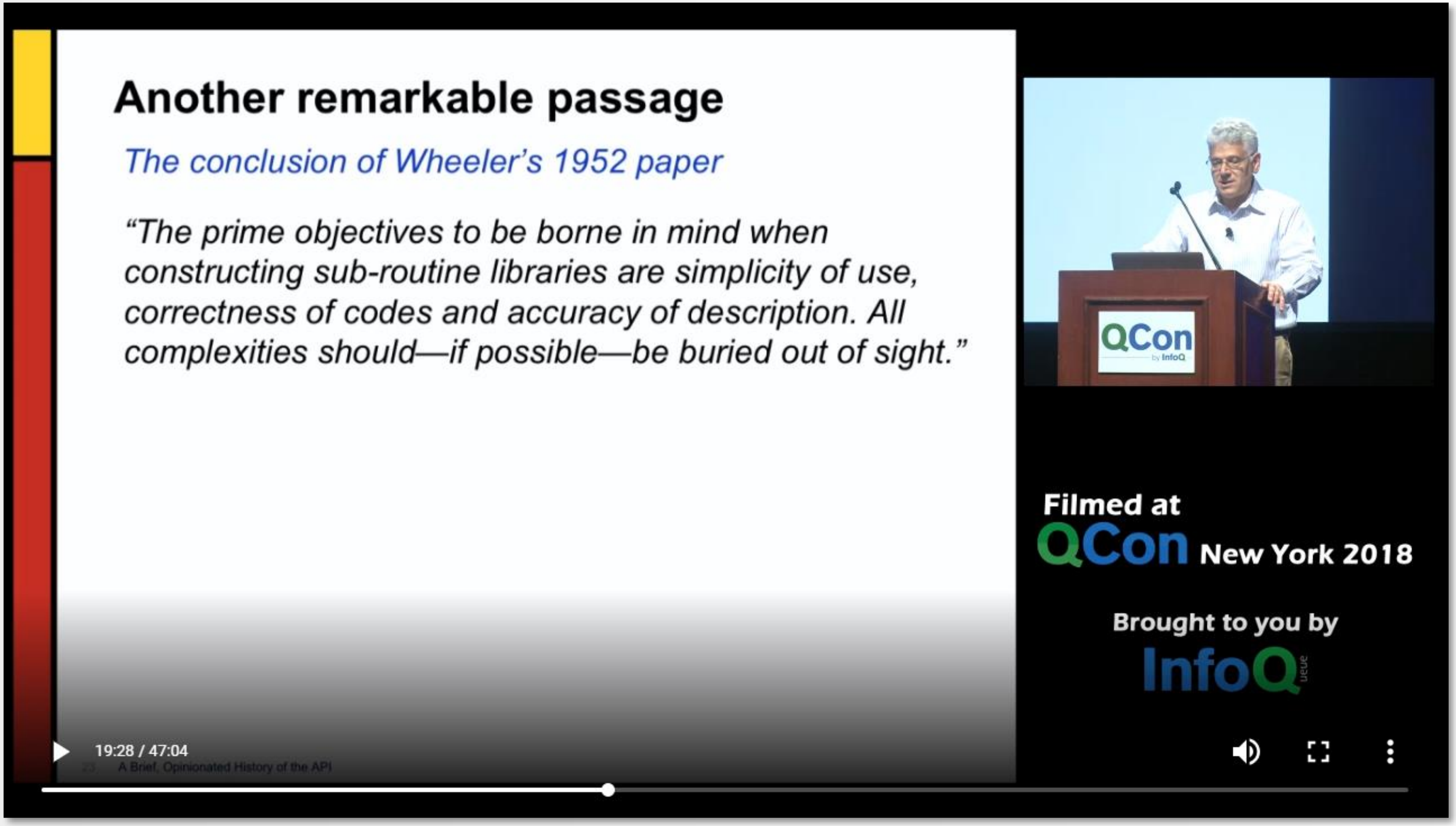
2. Simplicity – Why is it important

Complicated APIs are frustrating.

2. Simplicity – See also convenience

```
// Using jOOR - Convenience on top of JDK
// reflection and compilation APIs
Class<?> myClass = Reflect.compile(
    "com.example.MyClass",

    """
    package com.example;
    public class MyClass {
    }
    """
).type();
```

2. Simplicity – This isn't news



Another remarkable passage

The conclusion of Wheeler's 1952 paper

"The prime objectives to be borne in mind when constructing sub-routine libraries are simplicity of use, correctness of codes and accuracy of description. All complexities should—if possible—be buried out of sight."

Josh Bloch https://www.infoq.com/presentations/history-api

2. Simplicity – How to be simple

This is really difficult

2. Simplicity – How to be simple

Some things are «obvious»

… like low coupling

2. Simplicity – How to be simple

But simplicity is a lot of work.

2. Simplicity – How to be simple

But simplicity is a lot of work.

Simplicity emerges from very careful design.

2. Simplicity – How to be simple

But simplicity is a lot of work.

Simplicity emerges from very careful design.

And tons of iterations.

2. Simplicity – How to be simple

Simplicity is like obscenity

«I know it when I see it»

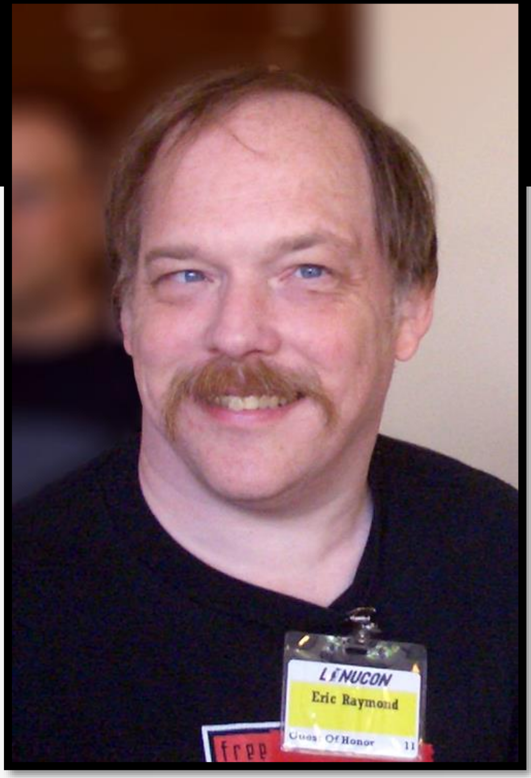
-- United States Supreme Court Justice Potter Stewart

3. Do One Thing

DOTADIW

Do One Thing and Do It Well.

3. Do One Thing

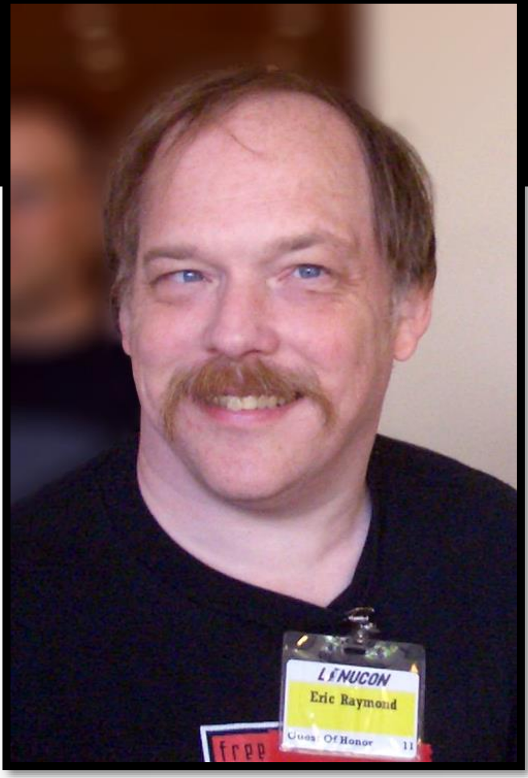


Eric Raymond's 17 Unix Rules

1. Build modular programs
2. Write readable programs
3. Use composition
4. Separate mechanisms from policy
5. Write simple programs
6. Write small programs
7. Write transparent programs
8. Write robust programs
9. Make data complicated when required, not the program
10. Build on potential users' expected knowledge
11. Avoid unnecessary output
12. Write programs which fail in a way easy to diagnose
13. Value developer time over machine time
14. Write abstract programs that generate code instead of writing code by hand
15. Prototype software before polishing it
16. Write flexible and open programs
17. Make the program and protocols extensible.

https://en.wikipedia.org/wiki/Unix_philosophy#Do_One_Thing_and_Do_It_Well
https://en.wikipedia.org/wiki/Eric_S._Raymond#/media/File:Eric_S_Raymond_portrait.jpg License CC BY-SA 2.0

3. Do One Thing



Eric Raymond's 17 Unix Rules

1. Build modular programs
2. Write readable programs
3. Use composition
4. Separate mechanisms from policy
5. Write simple programs
6. Write small programs
7. Write transparent programs
8. Write robust programs
9. Make data complicated when required, not the program
10. Build on potential users' expected knowledge
11. Avoid unnecessary output
12. Write programs which fail in a way easy to diagnose
13. Value developer time over machine time
14. Write abstract programs that generate code instead of writing by hand
15. Prototype software before polishing it
16. Write flexible and open programs
17. Make the program and protocols extensible.

Do One Thing

https://en.wikipedia.org/wiki/Unix_philosophy#Do_One_Thing_and_D
https://en.wikipedia.org/wiki/Eric_S._Raymond#/media/File:Eric_S_Ra

3. Do One Thing

You cannot implement simplicity without Do One Thing.

4. Types

https://www.amazon.co.uk/Imagination-Intelligence-Cognitive-Educational-Movements/dp/B07DL4WX2H

4. Types – Typesafety

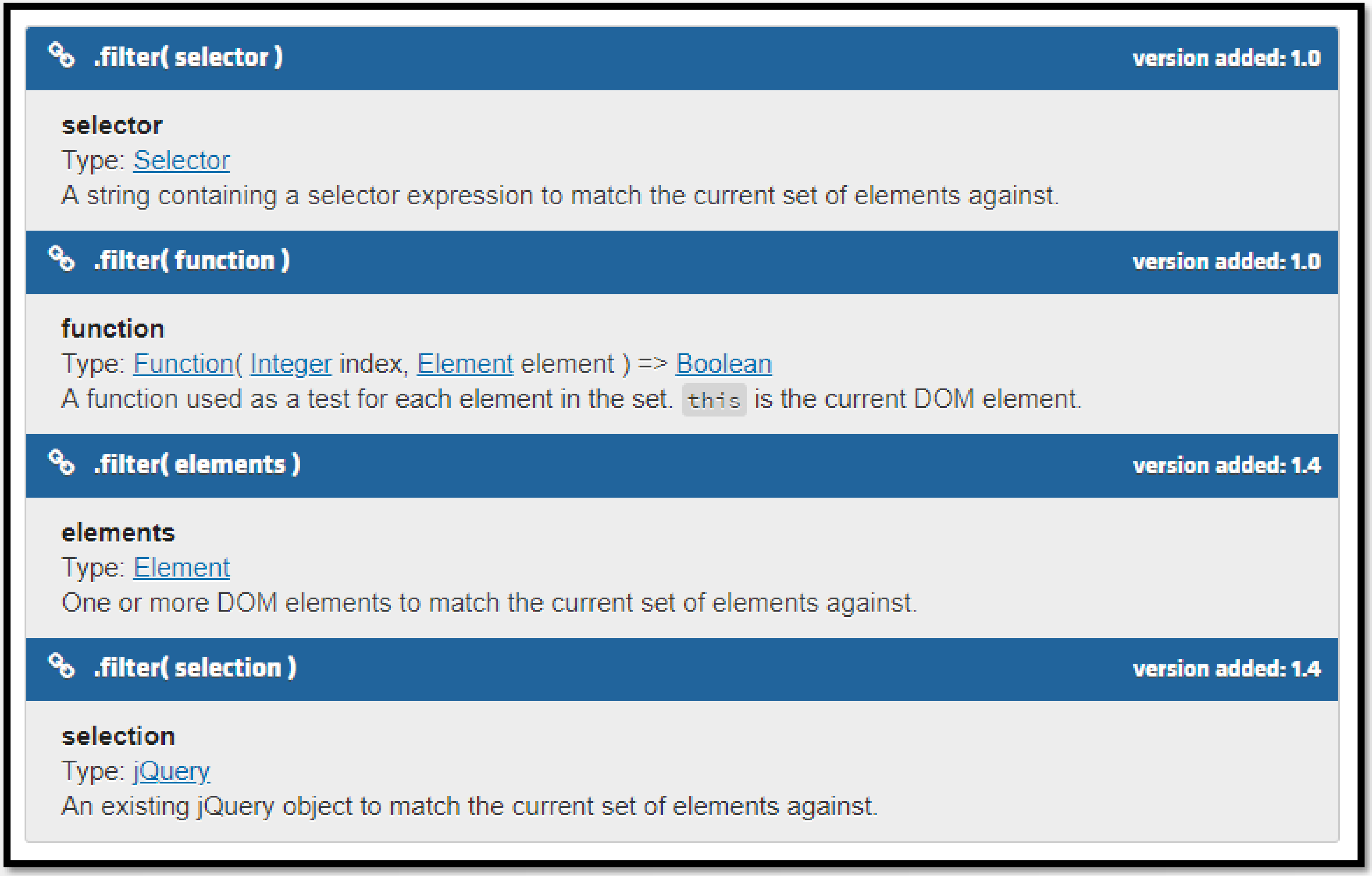
Type safety is one of those bikesheds

4. Types – Typesafety

Great APIs can exist without type safety

… but I doubt they exist without types

4. Types – Typesafety



.filter(selector) version added: 1.0

selector
Type: Selector
A string containing a selector expression to match the current set of elements against.

.filter(function) version added: 1.0

function
Type: Function(Integer index, Element element) => Boolean
A function used as a test for each element in the set. `this` is the current DOM element.

.filter(elements) version added: 1.4

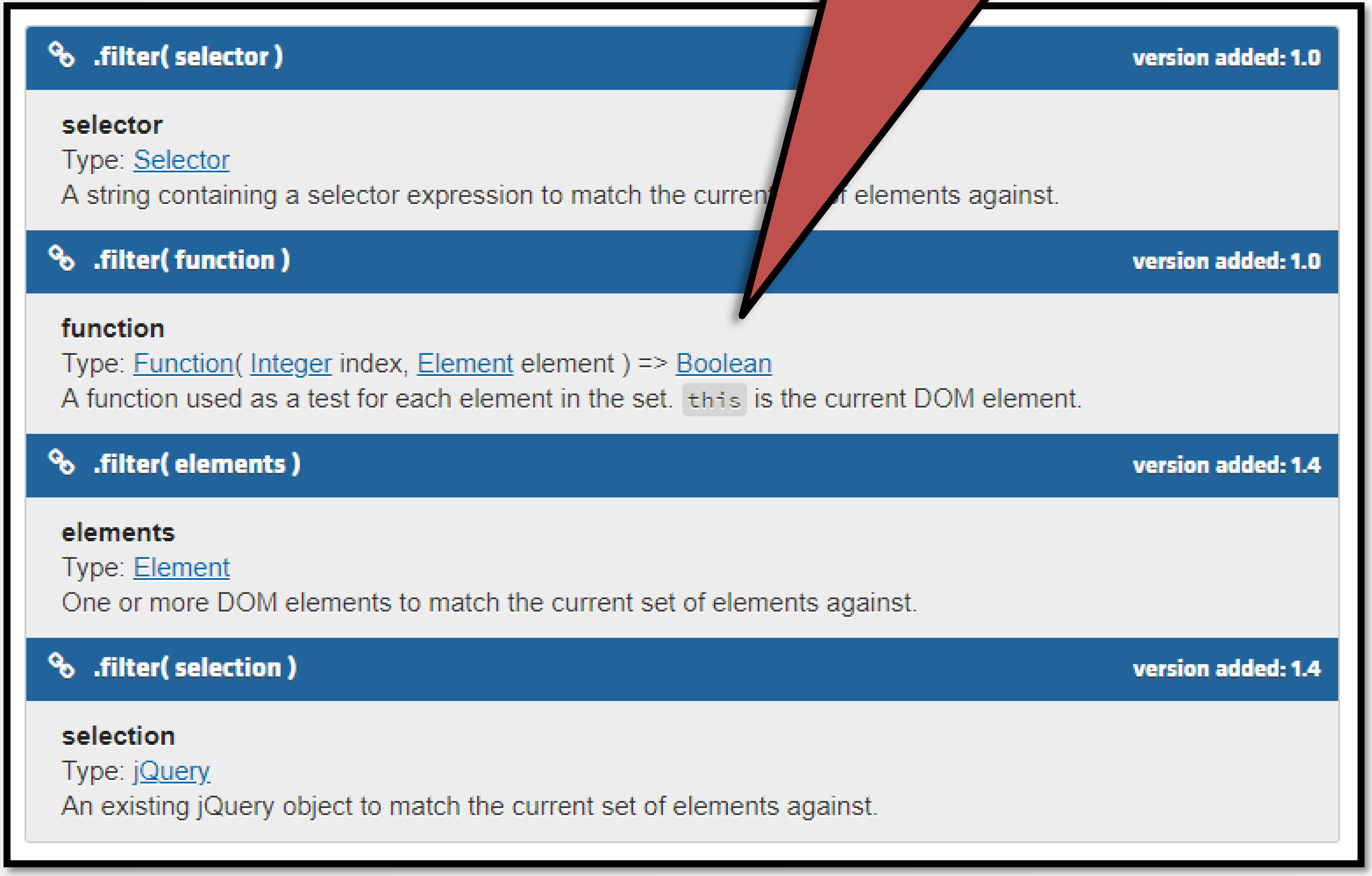
elements
Type: Element
One or more DOM elements to match the current set of elements against.

.filter(selection) version added: 1.4

selection
Type: jQuery
An existing jQuery object to match the current set of elements against.

4. Types – Typesafety

Types are in the docs



.filter(selector) version added: 1.0

selector
Type: Selector
A string containing a selector expression to match the curren f elements against.

.filter(function) version added: 1.0

function
Type: Function(Integer index, Element element) => Boolean
A function used as a test for each element in the set. `this` is the current DOM element.

.filter(elements) version added: 1.4

elements
Type: Element
One or more DOM elements to match the current set of elements against.

.filter(selection) version added: 1.4

selection
Type: jQuery
An existing jQuery object to match the current set of elements against.

4. Types – Advantages

You should design types regardless if you type check them.

4. Types – Advantages

Likewise there are no «schemaless» DBMS.

4. Types – Advantages

There are only «schema-on-read» and «schema-on-write» DBMS.

4. Types – Advantages

Types have a few decisive advantages

4. Types – Advantages

1. They (may) have a name
2. They simplify your design (if done well)
3. They do one thing
4. They’re types and thus type safe
5. They help discover the API
6. They describe errors
7. They can be applied consistently
8. They lead to better convenience
9. They can be versioned
10. They can be documented

4. Types – Recognise the outline of this talk?

10 Reasons Why we Love Some APIs and Why we Hate Some Others

1. Naming
2. Simplicity
3. Do One Thing
4. Types
5. Discoverability
6. Error Handling
7. Consistency
8. Convenience
9. Compatibility
10. Documentation

4. Types – R

In a way, this talk is about types

10 Reasons Why we Lov s and Why we Hate Some Others

1. Naming
2. Simplicity
3. Do One Thing
4. Types
5. Discoverability
6. Error Handling
7. Consistency
8. Convenience
9. Compatibility
10. Documentation

4. Types – How do we do types?

Some prefer nominal types

```
class J2eeBasedPreAuthenticatedWebAuthenticationDetailsSource

{ ... }
```

https://docs.spring.io/spring-security/site/docs/4.2.12.BUILD-SNAPSHOT/apidocs/org/springframework/security/web/authentication/preauth/j2ee/J2eeBasedPreAuthenticatedWebAuthenticationDetailsSource.html

4. Types – How do we do types?

refer nominal types

```
class J2eeBasedPreAuthenticatedWebAuthenticationDetailsSource
implements AuthenticationDetailsSource<HttpServletRequest,
  PreAuthenticatedGrantedAuthoritiesWebAuthenticationDetails>
{ ... }
```

https://docs.spring.io/spring-security/site/docs/4.2.12.BUILD-SNAPSHOT/apidocs/org/springframework/security/web/authentication/preauth/j2ee/J2eeBasedPreAuthenticatedWebAuthenticationDetailsSource.html

4. Types – How do we do types?

Some prefer structural types

```
public Mono<CollectionModel<EntityModel<Employee>>> all() { ... }
```

https://spring.io/blog/2019/03/05/spring-hateoas-1-0-m1-released

ypes?

uctural types

```
public Mono<CollectionModel<EntityModel<Employee>>> all() { ... }
```

Are you thinking what I’m thinking?

4. Types – How do we do types?

Some prefer structural types

```
public Mono<Publisher<Optional<CollectionModel<Stream<EntityModel<Try<
  Employee
>>>>>>> all() { ... }
```

Make no mistake: Not a merge conflicts

4. Types – How do we do types?

Some people add syntax sugar.

4. Types – How do we do types?

Some people add syntax sugar.

Some people add syntax vinegar.

4. Types – How do we do types?

So easy to make fun of others.

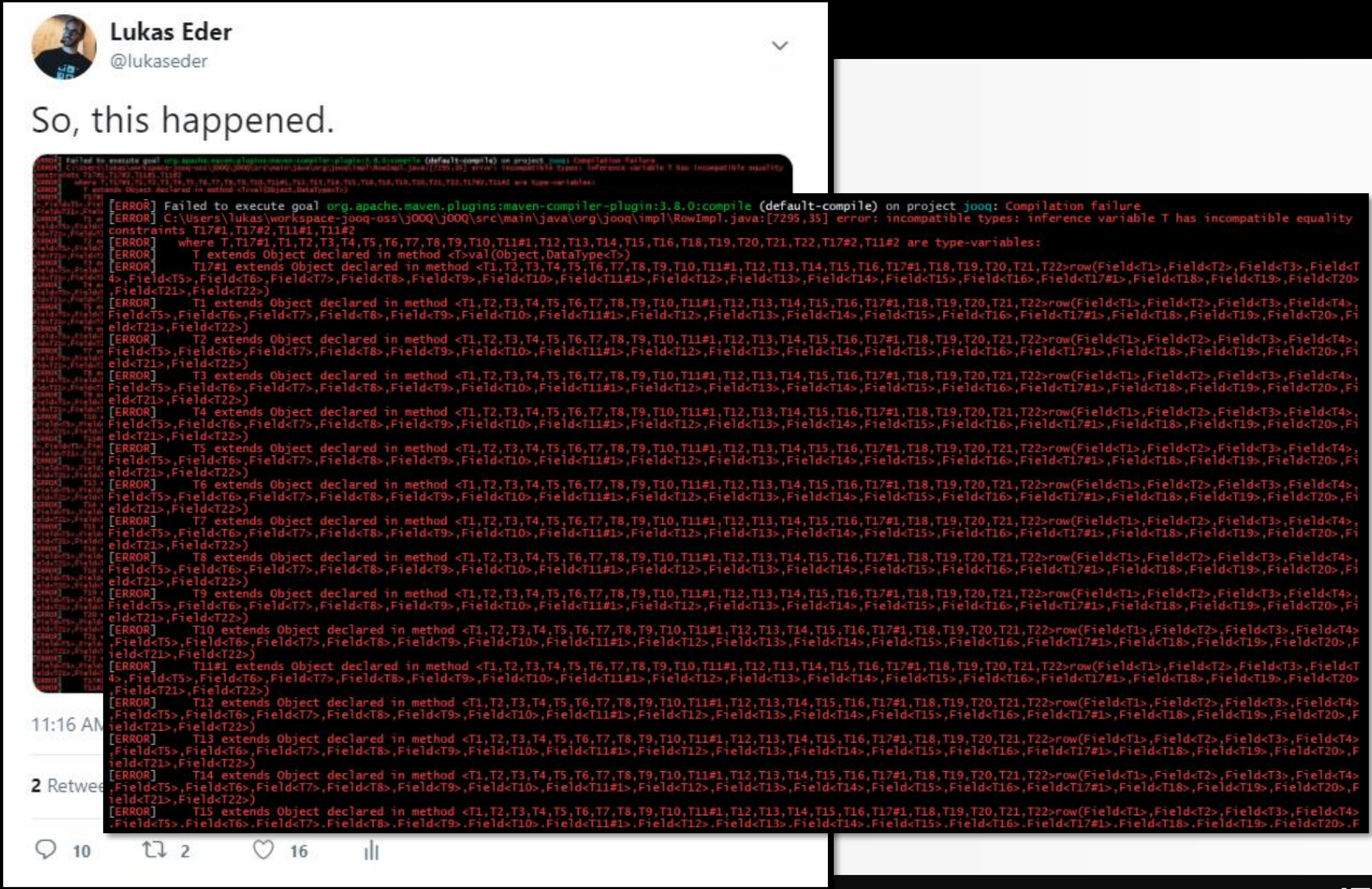
4. Types – How do we do types?

```
package org.jooq.impl;

public class DSL {
  public static <T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12,
    T13, T14, T15, T16, T17, T18, T19, T20, T21, T22> Row22<T1, T2,
    T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16,
    T17, T18, T19, T20, T21, T22> row(T1 t1, T2 t2, T3 t3, T4 t4,
    T5 t5, T6 t6, T7 t7, T8 t8, T9 t9, T10 t10, T11 t11, T12 t12,
    T13 t13, T14 t14, T15 t15, T16 t16, T17 t17, T18 t18, T19 t19,
    T20 t20, T21 t21, T22 t22) {
    ...
  }
}
```

4. Types – How do we do types?

```
package org.jooq.impl;

public class DSL {
  public static <T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12,
    T13, T14, T15, T16, T17, T18, T19, T20, T21, T22> Row22<T1, T2,
    T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16,
    T17, T18, T19, T20, T21, T22> row(T1 t1, T2 t2, T3 t3, T4 t4,
    T5 t5, T6 t6, T7 t7, T8 t8, T9 t9, T10 t10, T11 t11, T12 t12,
    T13 t13, T14 t14, T15 t15, T16 t16, T17 t17, T18 t18, T19 t19,
    T20 t20, T21 t21, T22 t22) {
    ...
  }
}
```

4. Types – How do we do types?



Lukas Eder
@lukaseder

So, this happened.

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.8.0:compile (default-compile) on project jooq: Compilation failure
[ERROR] C:\Users\lukas\workspace-jooq-oss\jOOQ\jOOQ\src\main\java\org\jooq\impl\RowImpl.java:[7295,35] error: incompatible types: inference variable T has incompatible equality
constraints T17#1,T17#2,T11#1,T11#2
[ERROR]   where T,T17#1,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T18,T19,T20,T21,T22,T17#2,T11#2 are type-variables:
[ERROR]     T extends Object declared in method <T>val(Object,DataType<T>)
[ERROR]     T17#1 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T
4>,Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>
,Field<T21>,Field<T22>)
[ERROR]     T1 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T2 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T3 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T4 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T5 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T6 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T7 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T8 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T9 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>,
Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,Fi
eld<T21>,Field<T22>)
[ERROR]     T10 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>
,Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,F
ield<T21>,Field<T22>)
[ERROR]     T11#1 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T
4>,Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>
,Field<T21>,Field<T22>)
[ERROR]     T12 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>
,Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,F
ield<T21>,Field<T22>)
[ERROR]     T13 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>
,Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,F
ield<T21>,Field<T22>)
[ERROR]     T14 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>
,Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,F
ield<T21>,Field<T22>)
[ERROR]     T15 extends Object declared in method <T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11#1,T12,T13,T14,T15,T16,T17#1,T18,T19,T20,T21,T22>row(Field<T1>,Field<T2>,Field<T3>,Field<T4>
,Field<T5>,Field<T6>,Field<T7>,Field<T8>,Field<T9>,Field<T10>,Field<T11#1>,Field<T12>,Field<T13>,Field<T14>,Field<T15>,Field<T16>,Field<T17#1>,Field<T18>,Field<T19>,Field<T20>,F
```

11:16 AM

2 Retwee

4. Types – How do we do types?

```
package org.jooq;

public interface Row22<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11,
  T12, T13, T14, T15, T16, T17, T18, T19, T20, T21, T22> extends Row {

  ...

  Condition eq(QuantifiedSelect<? extends Record22<T1, T2, T3, T4, T5,
    T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18, T19,
    T20, T21, T22>> select);
}
```

4. Types – How do we do types?

```
// You actually don’t see those types
Condition condition = row(1, "Lukas", "Eder").eq(any(
  select(
    SPEAKER.ID,
    SPEAKER.FIRST_NAME,
    SPEAKER.LAST_NAME
  )
  .from(SPEAKER)
));
```

4. Types – How do we do types?

```
// You actually don’t see those types ... until you get it wrong
Condition condition = row(1, "Lukas", "Eder").eq(any(
  select(
    SPEAKER.ID,
    SPEAKER.FIRST_NAME
  )
  .from(SPEAKER)
));
```

Compilation error

4. Types – How do we do types?

What’s my point?

4. Types – How do we do types?

Types are semantic

4. Types – How do we do types?

Types convey meaning

4. Types – How do we do types?

Types self-document

4. Types – How do we do types?

This works both with nominal *and* structural typing

4. Types – How do we do types?

This works both with static *and* dynamic typing

4. Types – Bad example

Counter example

4. Types – Bad example

```
package java.sql;

public interface Connection {
  Statement createStatement(
    int resultSetType,
    int resultSetConcurrency,
    int resultSetHoldability
  ) throws SQLException;
}
```

These things are defined in... ResultSet (╯°□°) ╯︵ ┻━┻

4. Types – Much better

```
package java.sql;

// This would have be so much better
public enum ResultSetType { ... }
public enum ResultSetConcurrency { ... }
public enum ResultSetHoldability { ... }

public interface Connection {
  Statement createStatement(
    ResultSetType resultSetType,
    ResultSetConcurrency resultSetConcurrency,
    ResultSetHoldability resultSetHoldability
  ) throws SQLException;
}
```

4. Types – Bad example: stringly typed

You know what else is stringly typed?

4. Types – Bad example: stringly typed

```
@Query("SELECT * FROM user")
```

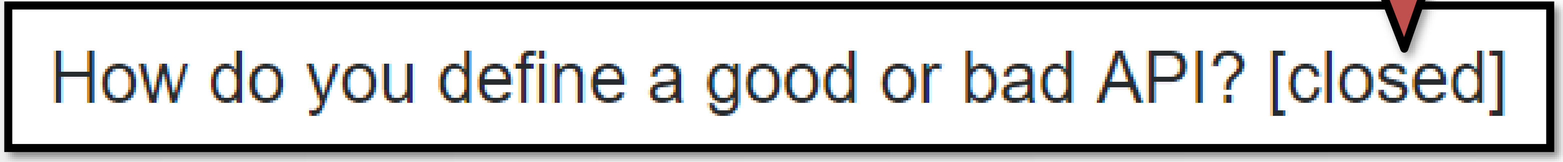
5. Discoverability

Picture by Valentin Antonucci (Pexels License) https://www.pexels.com/photo/person-holding-compass-691637/

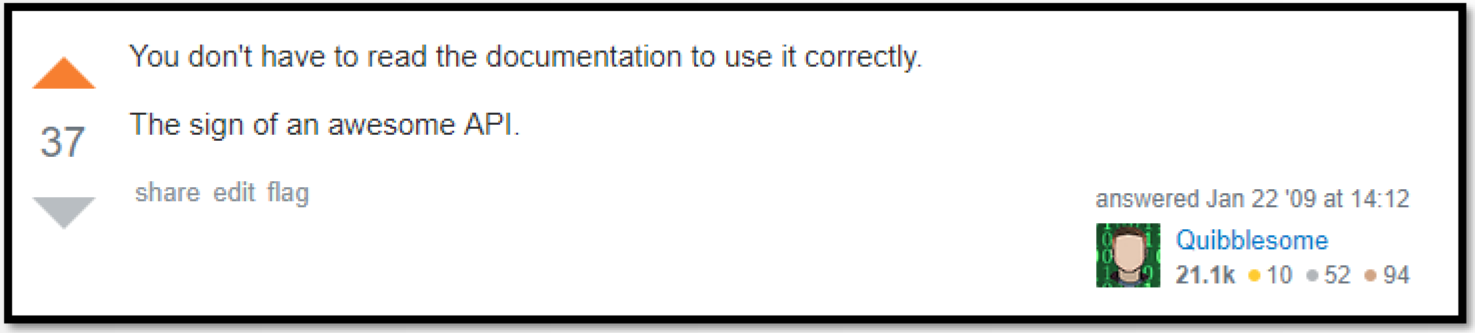
5. Discoverability

Good question. Closed, obviously

stackoverflow



How do you define a good or bad API? [closed]



37

You don't have to read the documentation to use it correctly.

The sign of an awesome API.

answered Jan 22 '09 at 14:12

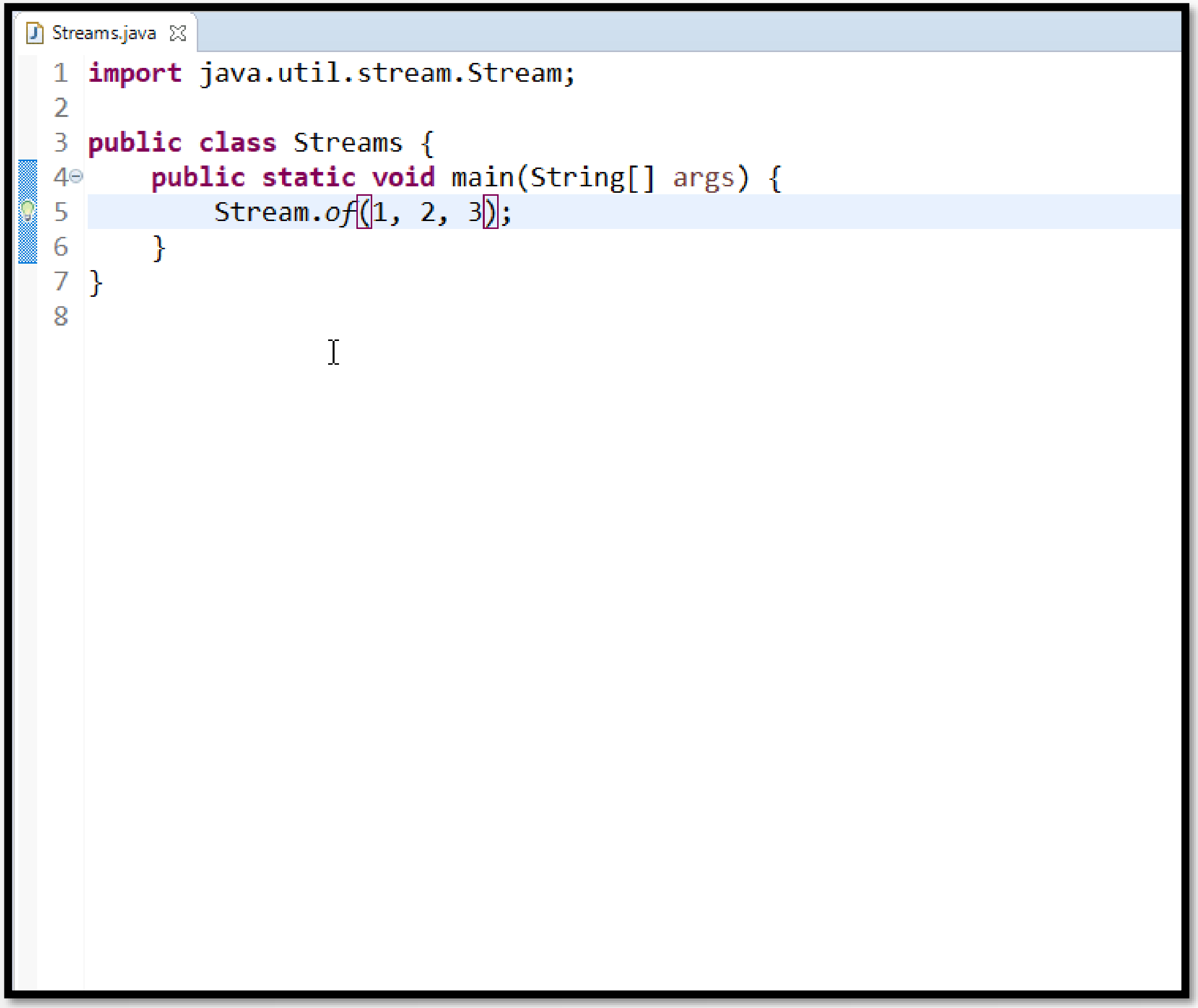
Quibblesome

21.1k ● 10 ● 52 ● 94

5. Discoverability

Do your users *have to* RTFM (read the docs)?

5. Discoverability

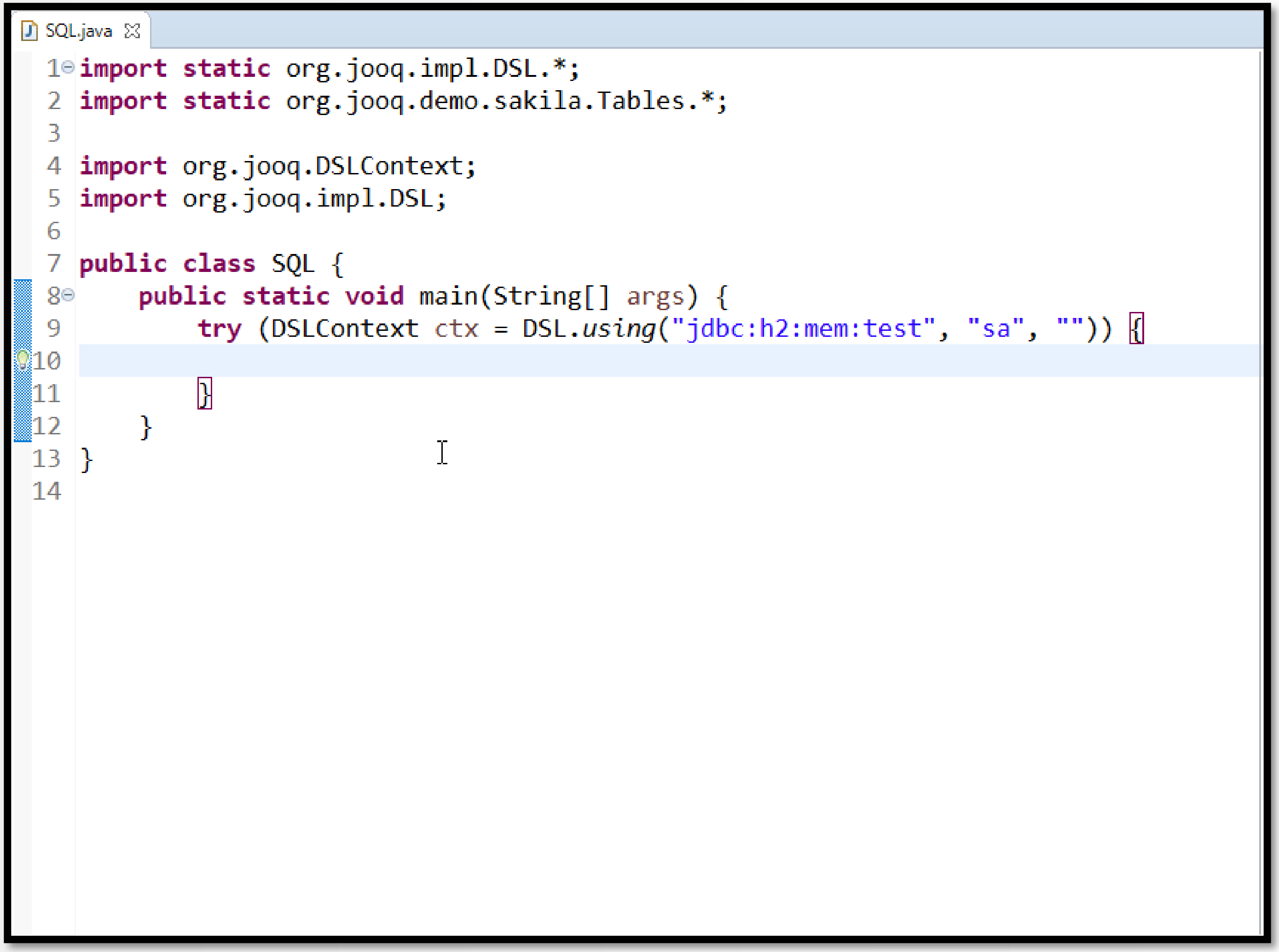


```
import java.util.stream.Stream;

public class Streams {
    public static void main(String[] args) {
        Stream.of(1, 2, 3);
    }
}
```

5. Discoverability

Another *obviously* great API

5. Discoverability



```java
import static org.jooq.impl.DSL.*;
import static org.jooq.demo.sakila.Tables.*;

import org.jooq.DSLContext;
import org.jooq.impl.DSL;

public class SQL {
    public static void main(String[] args) {
        try (DSLContext ctx = DSL.using("jdbc:h2:mem:test", "sa", "")) {

        }
    }
}
```

5. Discoverability

Some people like reading docs.

5. Discoverability

Some people like reading docs.

Others like navigating the API.

... can your API accommodate both?

5. Discoverability

An API should have a very small set of entry points

5. Discoverability – JPA good part

```
emTx((EntityManager em) -> {
  List<Film> films = em
    .createNativeQuery("SELECT * FROM film", Film.class)
    .getResultList();

  for (Film film : films)
    System.out.println(film.actors.size());
});
```

5.

API can be discovered from EntityManager

```
emTx((EntityManager em) -> {
  List<Film> films = em
    .createNativeQuery("SELECT * FROM film", Film.class)
    .getResultList();

  for (Film film : films)
    System.out.println(film.actors.size());
});
```

5. Discoverability – JPA good part

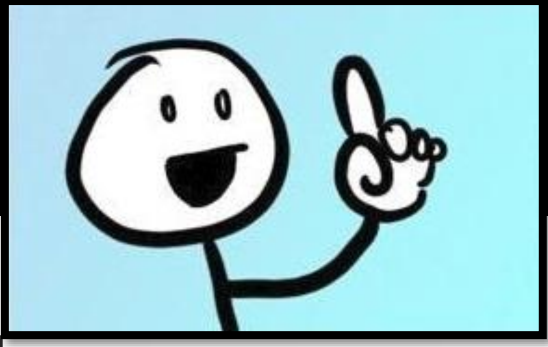
```
emTx((EntityManager em) -> {
  List<Film> films = em
    .createNativeQuery("SELECT * FROM f     , Film.class)
    .getResultList();

  for (Film film : films)
    System.out.println(film.actors.size());
});
```

5. Discoverability – JPA good part

```
emTx((EntityManager em) -> {
  List<Film> films = em
    .createNativeQuery("SELECT * FROM film", Film.class)
    .getResultList();

  for (Film film : films)
    System.out.println(film.actors.size());
});
```

But I'm interested in this

5. Discoverability

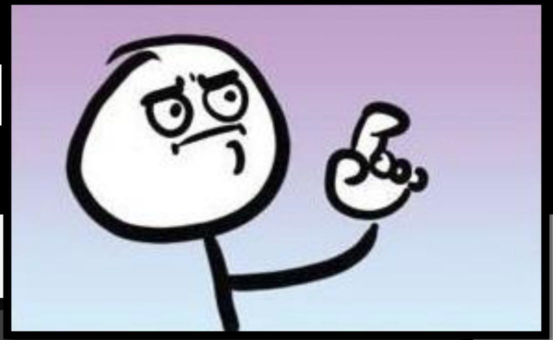


What I wanted

```
void emTx(Consumer<EntityManager> consumer) {
    consumer.accept(Tool.entityManager(dataSource));
}
```

In a way, EntityManager = DataSource

5. Discoverability – Bad examples (I

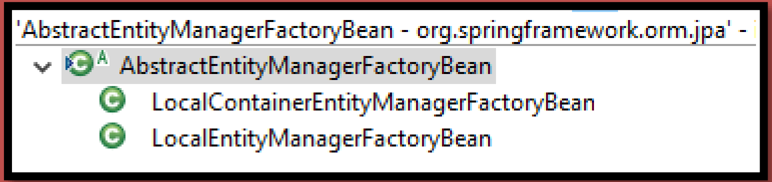


```java
void emTx(Consumer<EntityManager> consumer) {
    LocalContainerEntityManagerFactoryBean bean = new LocalContainerEntityManagerFactoryBean();
    HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
    adapter.setDatabasePlatform("org.hibernate.dialect.Oracle12cDialect");

    bean.setDataSource(datasource);
    bean.setPackagesToScan("com.examples.entities");
    bean.setJpaVendorAdapter(adapter);
    bean.setPersistenceUnitName("test");
    bean.setPersistenceProviderClass(HibernatePersistenceProvider.class);
    bean.afterPropertiesSet();

    EntityManagerFactory emf = bean.getObject();
    EntityManager em = emf.createEntityManager();

    em.getTransaction().begin();

    // Assume exception handling here
    consumer.accept(em);
    em.getTransaction().commit();
}
```

5. Discoverability – Bad examples (IMO)

```
void emTx(Consumer<EntityManager> consumer) {
    LocalContainerEntityManagerFactoryBean bean = new LocalContainerEntityManagerFactoryBean();
    HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
    adapter.setDatabasePlatform("org.hibernate.dialect.Oracle12cDialec  ");

    bean.setDataSource(datasource);
    bean.setPackagesToScan("com.examples.entities");
    bean.setJpaVendorAdapter(adapter);
    bean.setPersistenceUnitName("test");
    bean.setPersistenceProviderClass(HibernatePersistenceProvider.cla
    bean.afterPropertiesSet();

    EntityManagerFactory emf = bean.getObject();
    EntityManager em = emf.createEntityManager();

    em.getTransaction().

    // Assume exception
    consumer.accept(em);
    em.getTransaction().
}
```

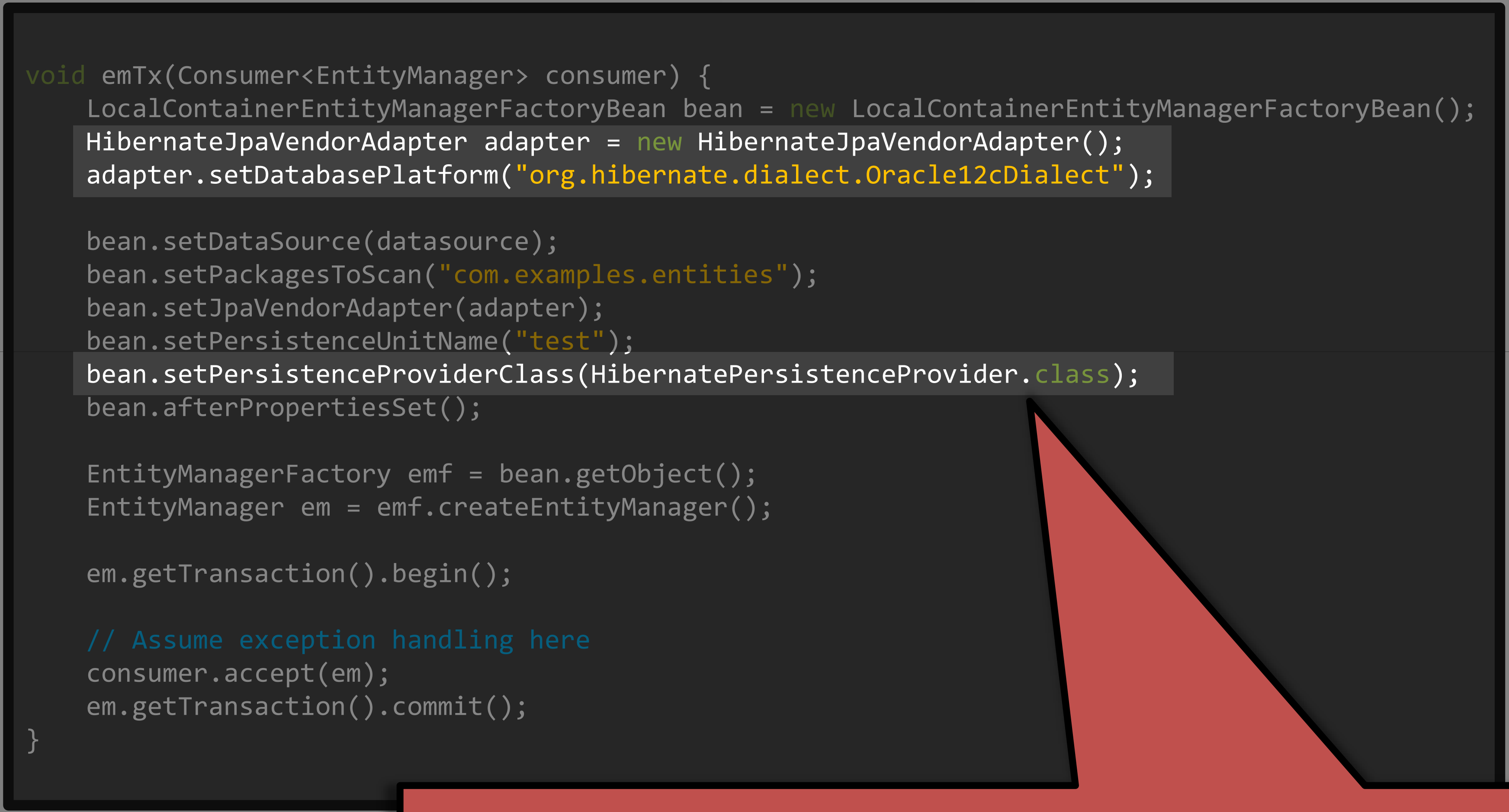
Guilty: I picked this at random



'AbstractEntityManagerFactoryBean - org.springframework.orm.jpa' -

- AbstractEntityManagerFactoryBean
  - LocalContainerEntityManagerFactoryBean
  - LocalEntityManagerFactoryBean

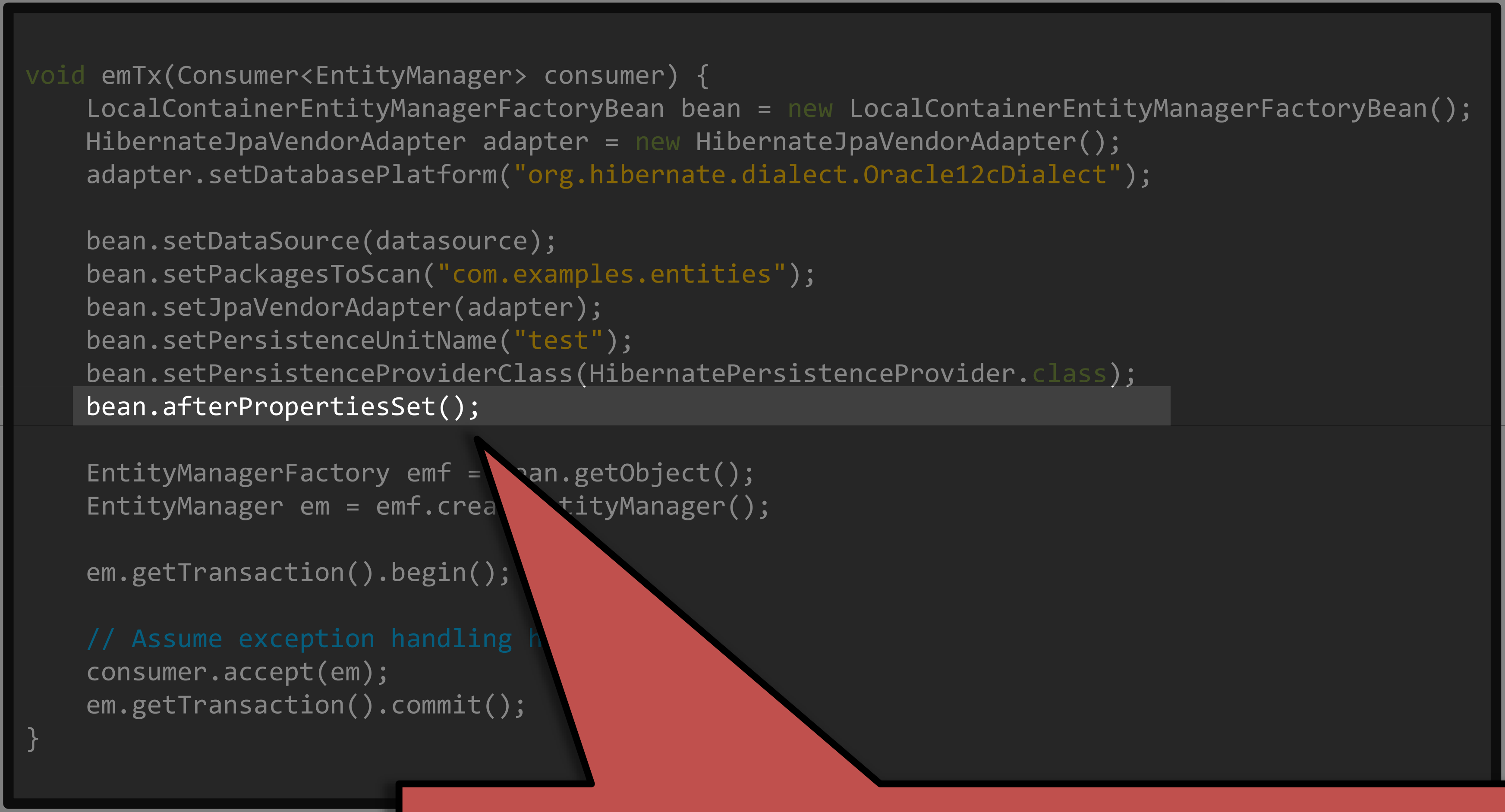
5. Discoverability – Bad examples (IMO)



```java
void emTx(Consumer<EntityManager> consumer) {
    LocalContainerEntityManagerFactoryBean bean = new LocalContainerEntityManagerFactoryBean();
    HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
    adapter.setDatabasePlatform("org.hibernate.dialect.Oracle12cDialect");

    bean.setDataSource(datasource);
    bean.setPackagesToScan("com.examples.entities");
    bean.setJpaVendorAdapter(adapter);
    bean.setPersistenceUnitName("test");
    bean.setPersistenceProviderClass(HibernatePersistenceProvider.class);
    bean.afterPropertiesSet();

    EntityManagerFactory emf = bean.getObject();
    EntityManager em = emf.createEntityManager();

    em.getTransaction().begin();

    // Assume exception handling here
    consumer.accept(em);
    em.getTransaction().commit();
}
```

I had to say «Hibernate» 3 times

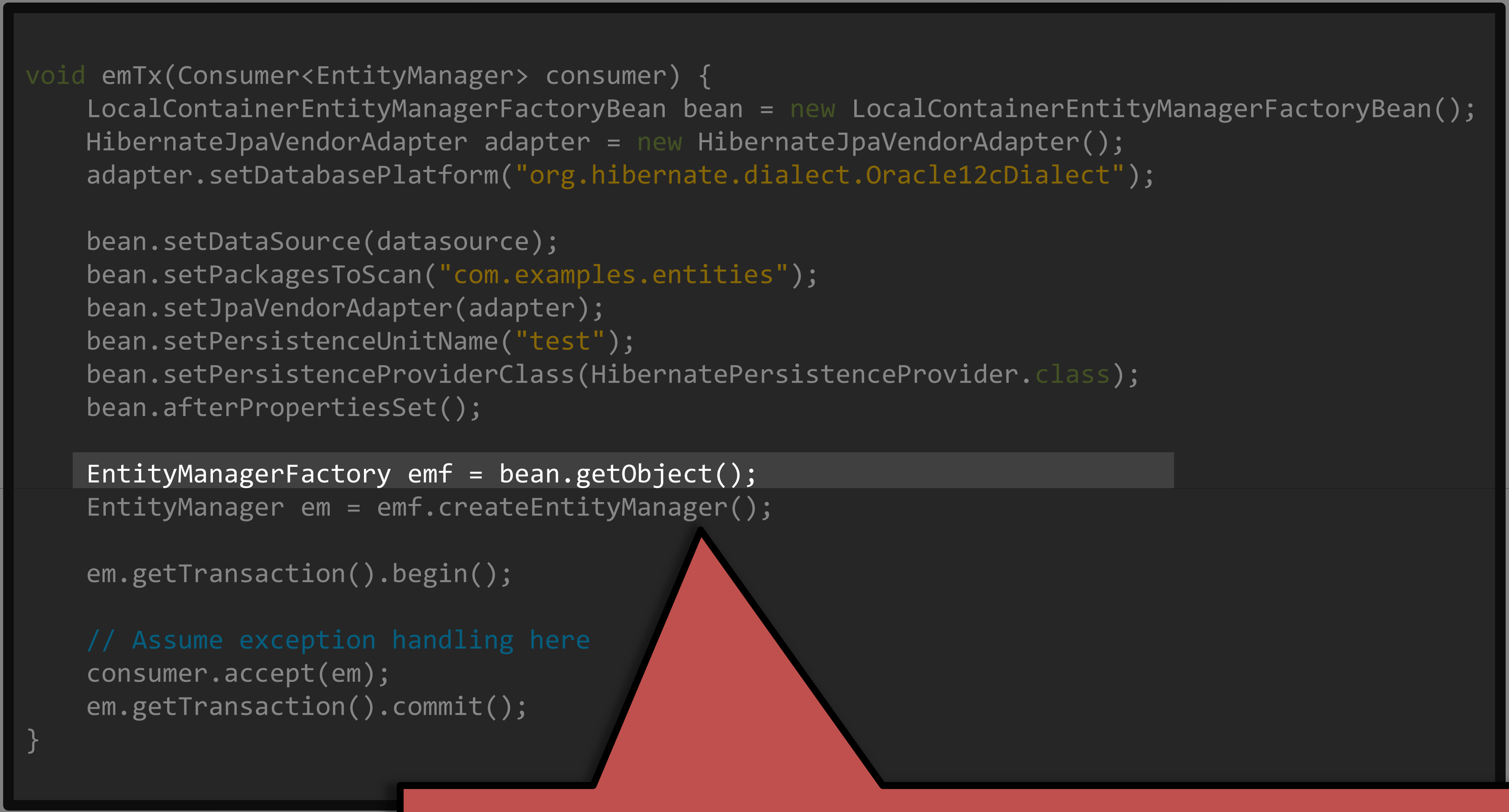
5. Discoverability – Bad examples (IMO)



```
void emTx(Consumer<EntityManager> consumer) {
    LocalContainerEntityManagerFactoryBean bean = new LocalContainerEntityManagerFactoryBean();
    HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
    adapter.setDatabasePlatform("org.hibernate.dialect.Oracle12cDialect");

    bean.setDataSource(datasource);
    bean.setPackagesToScan("com.examples.entities");
    bean.setJpaVendorAdapter(adapter);
    bean.setPersistenceUnitName("test");
    bean.setPersistenceProviderClass(HibernatePersistenceProvider.class);
    bean.afterPropertiesSet();

    EntityManagerFactory emf =   ean.getObject();
    EntityManager em = emf.crea   tityManager();

    em.getTransaction().begin();

    // Assume exception handling h
    consumer.accept(em);
    em.getTransaction().commit();
}
```

What on earth is this?

5. Discoverability – Bad examples (IMO)



```
void emTx(Consumer<EntityManager> consumer) {
    LocalContainerEntityManagerFactoryBean bean = new LocalContainerEntityManagerFactoryBean();
    HibernateJpaVendorAdapter adapter = new HibernateJpaVendorAdapter();
    adapter.setDatabasePlatform("org.hibernate.dialect.Oracle12cDialect");

    bean.setDataSource(datasource);
    bean.setPackagesToScan("com.examples.entities");
    bean.setJpaVendorAdapter(adapter);
    bean.setPersistenceUnitName("test");
    bean.setPersistenceProviderClass(HibernatePersistenceProvider.class);
    bean.afterPropertiesSet();

    EntityManagerFactory emf = bean.getObject();
    EntityManager em = emf.createEntityManager();

    em.getTransaction().begin();

    // Assume exception handling here
    consumer.accept(em);
    em.getTransaction().commit();
}
```

And why is it «getObject()»?

5. Discoverability

5. Discoverability

How long do you think it took me to discover this?

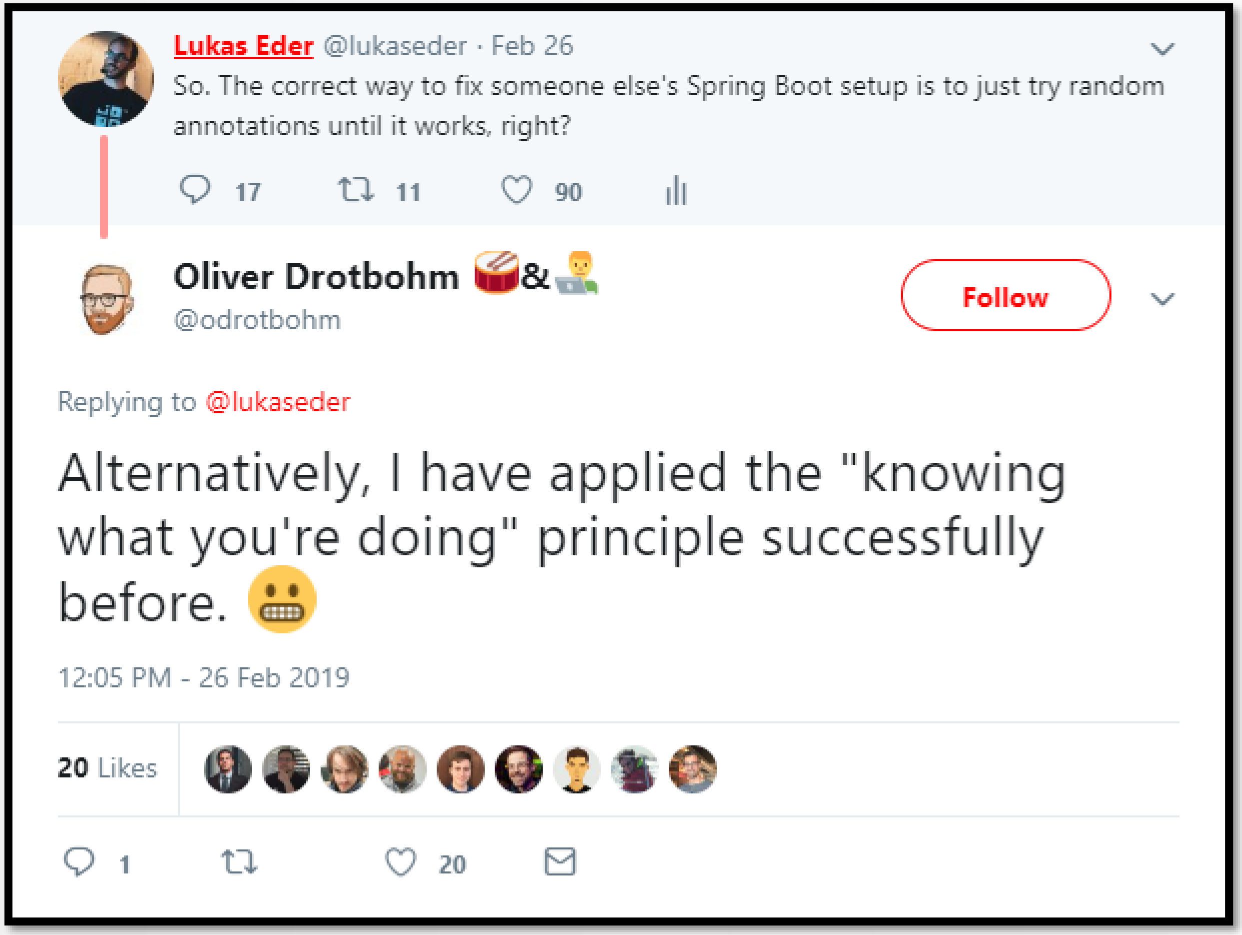
5. Discoverability

Do I have confidence that I’m doing it right?

5. Discoverability

I sure hope Olli will not be mad at me for quoting this ☺

5. Discoverability



Lukas Eder @lukaseder · Feb 26

So. The correct way to fix someone else's Spring Boot setup is to just try random annotations until it works, right?

17 11 90

Oliver Drotbohm 🥁&👨‍💻 @odrotbohm

Follow

Replying to @lukaseder

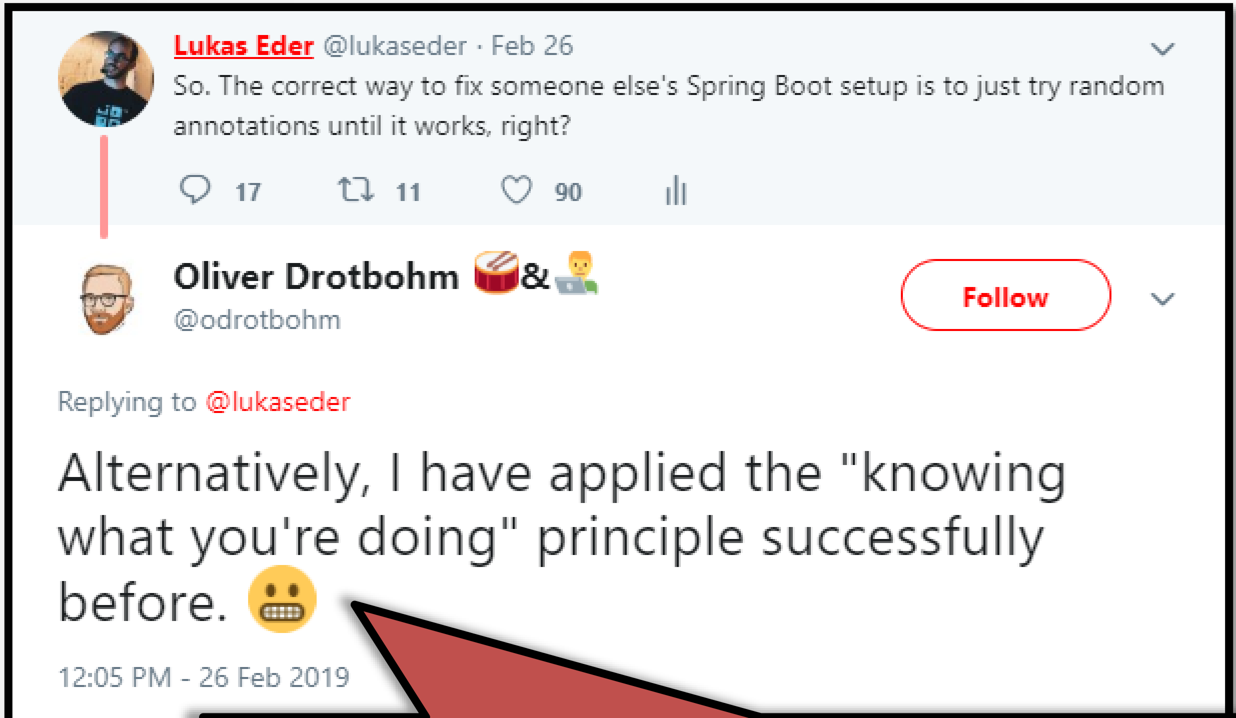
Alternatively, I have applied the "knowing what you're doing" principle successfully before. 😬

12:05 PM - 26 Feb 2019

20 Likes

1 20

5. Discoverability



Lukas Eder @lukaseder · Feb 26

So. The correct way to fix someone else's Spring Boot setup is to just try random annotations until it works, right?

Oliver Drotbohm 🥁&👨‍💻 @odrotbohm

Replying to @lukaseder

Alternatively, I have applied the "knowing what you're doing" principle successfully before. 😬

12:05 PM - 26 Feb 2019

20 Likes

Was I just RTFM'ed?

5. Discoverability

I admit: I don't know what I'm doing.

... and I don't feel bad about it

5. Discoverability

Docs are great.

5. Discoverability

Docs are great.

Discoverability is better.

6. Error Handling

6. Error Handling

```
catch (Exception e) {
  // Should never happen
}
```

7. Consistency

7. Consistency

Just kidding…

6. Error Handling

```
-- PL/SQL
begin


end;
```

6. Err

```
--
begin

end;
```

```
ORA-06550: line 3, column 1:
PLS-00103: Encountered the symbol "END" when expecting one of the following:

   ( begin case declare exit for goto if loop mod null pragma
   raise return select update while with <an identifier>
   <a double-quoted delimited-identifier> <a bind variable> <<
   continue close current delete fetch lock insert open rollback
   savepoint set sql execute commit forall merge pipe purge
   json_exists json_value json_query json_object json_array
06550. 00000 -  "line %s, column %s:\n%s"
*Cause:    Usually a PL/SQL compilation error.
*Action:
```

6. Error Handling

How about...

6. Error Handling

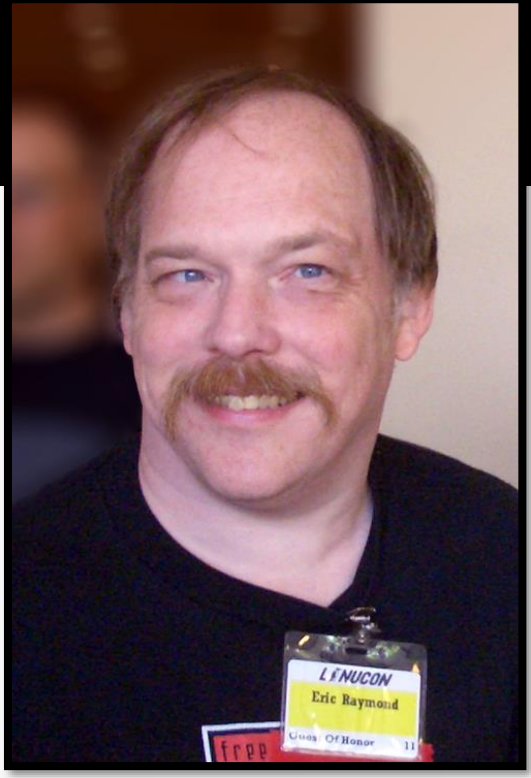
```
ORA-06550: line 3, column 1:
PLS-00103: block cannot be empty. At least one statement must be provided.
```

```
-- PL/ QL
begin


end;
```

6. Error Handling

```
-- PL/SQL
begin
  null;
end;
```

6. Error Handling

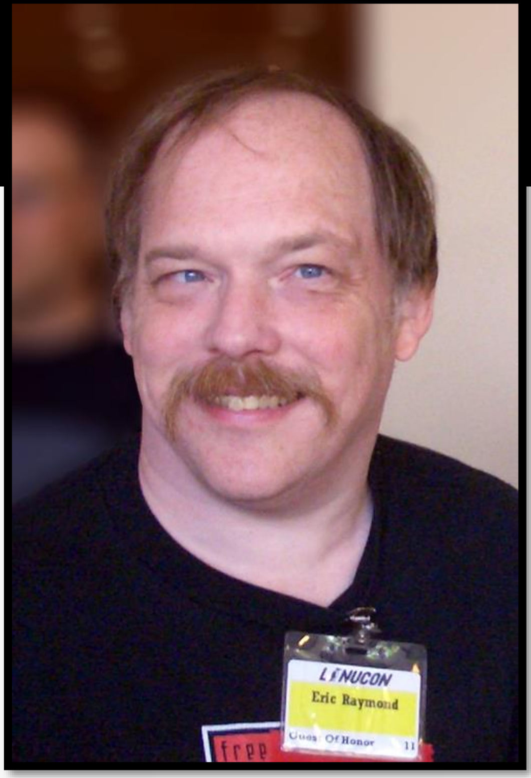


Eric Raymond's 17 Unix Rules

1. Build modular programs
2. Write readable programs
3. Use composition
4. Separate mechanisms from policy
5. Write simple programs
6. Write small programs
7. Write transparent programs
8. Write robust programs
9. Make data complicated when required, not the program
10. Build on potential users' expected knowledge
11. Avoid unnecessary output
12. Write programs which fail in a way easy to diagnose
13. Value developer time over machine time
14. Write abstract programs that generate code instead of writing code by hand
15. Prototype software before polishing it
16. Write flexible and open programs
17. Make the program and protocols extensible.

https://en.wikipedia.org/wiki/Unix_philosophy#Do_One_Thing_and_Do_It_Well
https://en.wikipedia.org/wiki/Eric_S._Raymond#/media/File:Eric_S_Raymond_portrait.jpg License CC BY-SA 2.0

6. Error Handling



Eric Raymond's 17 Unix Rules

1. Build modular programs
2. Write readable programs
3. Use composition
4. Separate mechanisms from policy
5. Write simple programs
6. Write small programs
7. Write transparent programs
8. Write robust programs
9. Make data complicated when required, not the program
10. Build on potential users' expected knowledge
11. Avoid unnecessary output
12. Write programs which fail in a way easy to diagnose
13. Value developer time over machine time
14. Write abstract programs that generate code instead of writing by hand
15. Prototype software before polishing it
16. Write flexible and open programs
17. Make the program and protocols extensible.

Error Handling

https://en.wikipedia.org/wiki/Unix_philosophy#Do_One_Thing_and_D
https://en.wikipedia.org/wiki/Eric_S._Raymond#/media/File:Eric_S_Ra

6. Error Handling

Picture by Jeremy Segrot (CC BY 2.0) https://www.flickr.com/photos/126337928@N05/42791980705

6. Error Handling

```
// c -- return value
int routine(...);
```

6. Error Handling

```
// c -- return value
int routine(...);

int err = routine(...);
if (err > 0) {
  // TODO should never happen
}
```

6. Error Handling

```
// Java -- (checked) exceptions
void routine(...) throws
  IllegalAccessException,
  IllegalArgumentException,
  InvocationTargetException
```

6. Error Handling

```
// Java -- (checked) exceptions
void routine(...) throws
  IllegalAccessException,
  IllegalArgumentException,
  InvocationTargetException

try {
  routine(...);
}
catch (InvocationTargetException ignore) {}
catch (IllegalArgumentException yolo) {}
catch (IllegalAccessException heh) {
  // That'll teach them
  System.exit(-1);
}
```

6. Error Handling

```
// Java -- Cool kids who know Scala
Either<Void, Error> routine(...);
```

6. Error Handling

```
// Java -- Cool kids who know Scala
Either<Void, Error> routine(...);

Error error = routine(...)
  .filter(r -> true)
  .get()
  .flatMap(r -> Either.right(r))
  .fold(l -> null, r -> r);

if (error != null) {
  // TODO should never happen
}
```

6. Error Handling

Mostly irrelevant

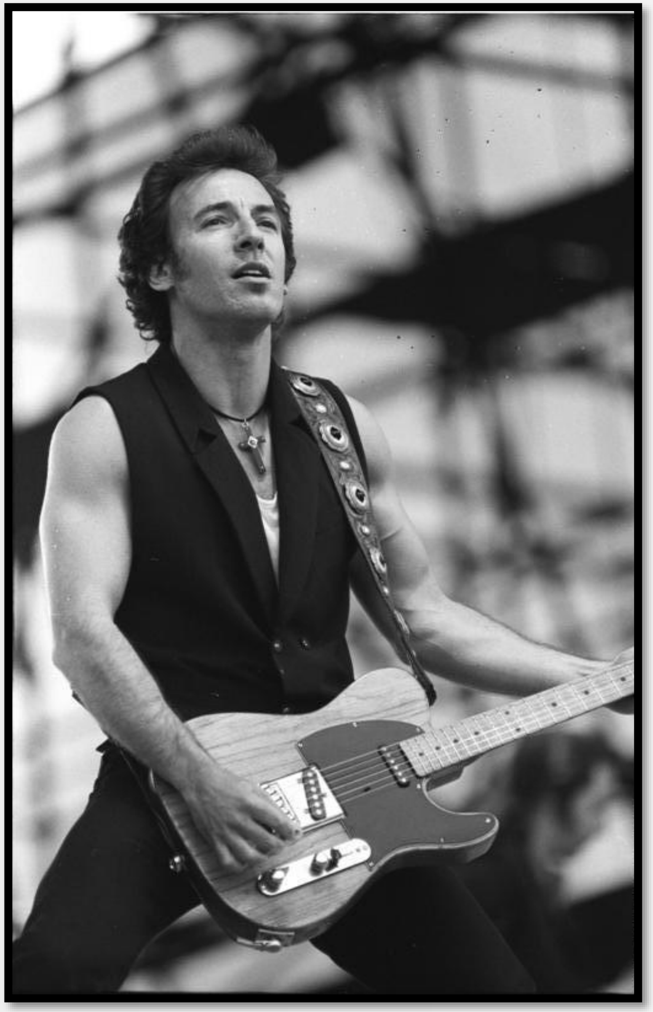
6. Error Handling

Be consistent and return meaningful errors. The caller should know what to do.

6. Error Handling

Types (Exceptions, Try monad, Either monad, etc.) are better than Strings / ints (error codes)

7. Consistency



> Getting an audience is hard. Sustaining an audience is hard. It demands a consistency of thought, of purpose, and of action over a long period of time.

—Bruce Springsteen

(Image: Bundesarchiv, Bild 183-1988-0719-38 / Uhlemann, Thomas / CC-BY-SA 3.0)

7. Consistency

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?
>
> Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
>
> Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
>
> Satisfaction: How pleasant is it to use the design?

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

7. Consistency

```java
// Average enterprise bean
public interface MyRepository {
  Customer[] getCustomers();
  List<Customer> getCustomersList();
  Stream<Customer> getCustomersByName(String name);
  Optional<Customer> getCustomerById(Integer id);
  Customer getCustomerBySocialSecurityNumber(String no);
  Customer getAnyCustomerByName(String name)
    throws ObjectNotFoundException;
}
```

7. Consistency

```
// Average enterprise bean
@Component @Bean
public interface MyRepository {
  Customer[] getCustomers();
  List<Customer> getCustomersList();
  Stream<Customer> getCustomersByName(String name);
  Optional<Customer> getCustomerById(Integer id);
  Customer getCustomerBySocialSecurityNumber(String no);
  Customer getAnyCustomerByName(String name)
    throws ObjectNotFoundException;
}
```

7. Consistency

```
// Average enterprise bean
@Component @Bean
@Discoverable
@AutoProxiable
public interface MyRepository {
  @Query("SELECT * FROM customers")
  Customer[] getCustomers();
  List<Customer> getCustomersList();

  @AutoClosingStreamableDevice
  Stream<Customer> getCustomersByName(String name);
  Optional<Customer> getCustomerById(Integer id);
  @POST @GET
  Customer getCustomerBySocialSecurityNumber(String no);
  Customer getAnyCustomerByName(String name)
    throws ObjectNotFoundException;
}
```

7. Consistency

```java
// Average enterprise bean
@Component @Bean
@Discoverable
@AutoProxiable
@Sendable @TODO @JIRAIssues ({1234, 81371, 617837})
@AutoFetchProxyThing(because=@ICan)
public interface MyRepository {
  @Query("SELECT * FROM customers")
  Customer @NonNull [] getCustomers();
  List<@NonNull Customer> getCustomersList();

  @AutoClosingStreamableDevice
  Stream<@NonNull Customer> getCustomersByName(@NonNull String name);
  Optional<@NonNull Customer> getCustomerById(@NonNull Integer id);
  @POST @GET
  Customer getCustomerBySocialSecurityNumber(@NonNull String no);
  Customer getAnyCustomerByName(@Nullable String name)
    throws ObjectNotFoundException;
}
```

7. Consistency

```
// Average enterprise bean
@Component @Bean
@Discoverable
@AutoProxiable
@Sendable @TODO @JIRAIssues ({1234, 81371, 617837})
@AutoFetchProxyThing(because=@ICan) @Bean
@Configuration
@NoArgsConstructor @Cloneable @SneakyThrows
@ResultSetMapping @ConcurrentInitialiserProxyFactory
@SpringPrefetchAdapter @AdapterProxyBeanMethod @AdapterBeanProxyMethod
@MoreAndMoreAnnotations @CanYouEvenReadThis
@IsThereStillAnyRealLogicLeft
public interface MyRepository {
  @Query("SELECT * FROM customers")
  Customer @NonNull [] getCustomers();
  List<@NonNull Customer> getCustomersList();
  @Bean
  @Configuration
  @NoArgsConstructor @Cloneable @SneakyThrows
  @ResultSetMapping @ConcurrentInitialiserProxyFactory
  @SpringPrefetchAdapter @AdapterProxyBeanMethod @AdapterBeanProxyMethod
  @MoreAndMoreAnnotations @CanYouEvenReadThis
  @IsThereStillAnyRealLogicLeft
  @AutoClosingStreamableDevice
  Stream<@NonNull Customer> getCustomersByName(@NonNull String name);
  Optional<@NonNull Customer> getCustomerById(@NonNull Integer id);
  @POST @GET
  Customer getCustomerBySocialSecurityNumber(@NonNull String no);
  Customer getAnyCustomerByName(@Nullable String name)
    throws ObjectNotFoundException;
}
```

7. Consistency

Me with this joke

COMING UP WITH AN ORIGINAL JOKE
RECYCLING THE SAME HUMOR IN A DIFFERENT WAY
vevo
memegenerator.net

7. Consistency

```
// Average ent
public interfa
  Customer[] g
  List<Custome
  Stream<Custo                             g name);
  Optional<Cus                             r id);
  Customer get                             r(String no);
  Customer get                             )
    throws Obj
}
```

7. Consistency

```
// Average enterprise bean
public interface MyRepository {
  Customer[] getCustomers();
  List<Customer> getCustomersList();
  Stream<Customer> getCustomersByName(String name);
  Optional<Customer> getCustomerById(Integer id);
  Customer getCustomerBySocialSecurityNumber(String no);
  Customer getAnyCustomerByName(String name)
    throws ObjectNotFoundException;
}
```

List vs Stream vs Array

Optional vs Null vs Exception

7. Consistency

All choices are fine (don’t bikeshed)

7. Consistency

All choices are fine (don’t bikeshed)

… but pick only one

7. Consistency

```
// Examples from some other language
// ---------------------------------

// Search $needle in $haystack (which is an array)
array_search ($needle, $haystack)

// In $haystack (which is a string), search $needle
strpos ($haystack, $needle)

// Search $search, replace by $replace in $subject
// (which is an array or a string)
str_replace ($search, $replace, $subject)
```

7. Consistency

If you’re not absolutely fluent, you have to look this up in the docs every time!

```
// Examples fro
// -------------

// Search $needle in $haystack (which is an
array_search ($needle, $haystack)

// In $haystack (which is a string), search $needle
strpos ($haystack, $needle)

// Search $search, replace by $replace
// (which is an array or a string)
str_replace ($search, $replace, $subje
```

7. Consistency

```
// Examples from some other language
// ---------------------------------

// Filter an $array using a $callback
array_filter ($array, $callback)

// Using a $callback, map an $array
array_map ($callback, $array)
```

7. Consistency

Why consistency?

7. Consistency

I’ve already mentioned JSR-310 as a good example for consistent naming and typing

7. Consistency

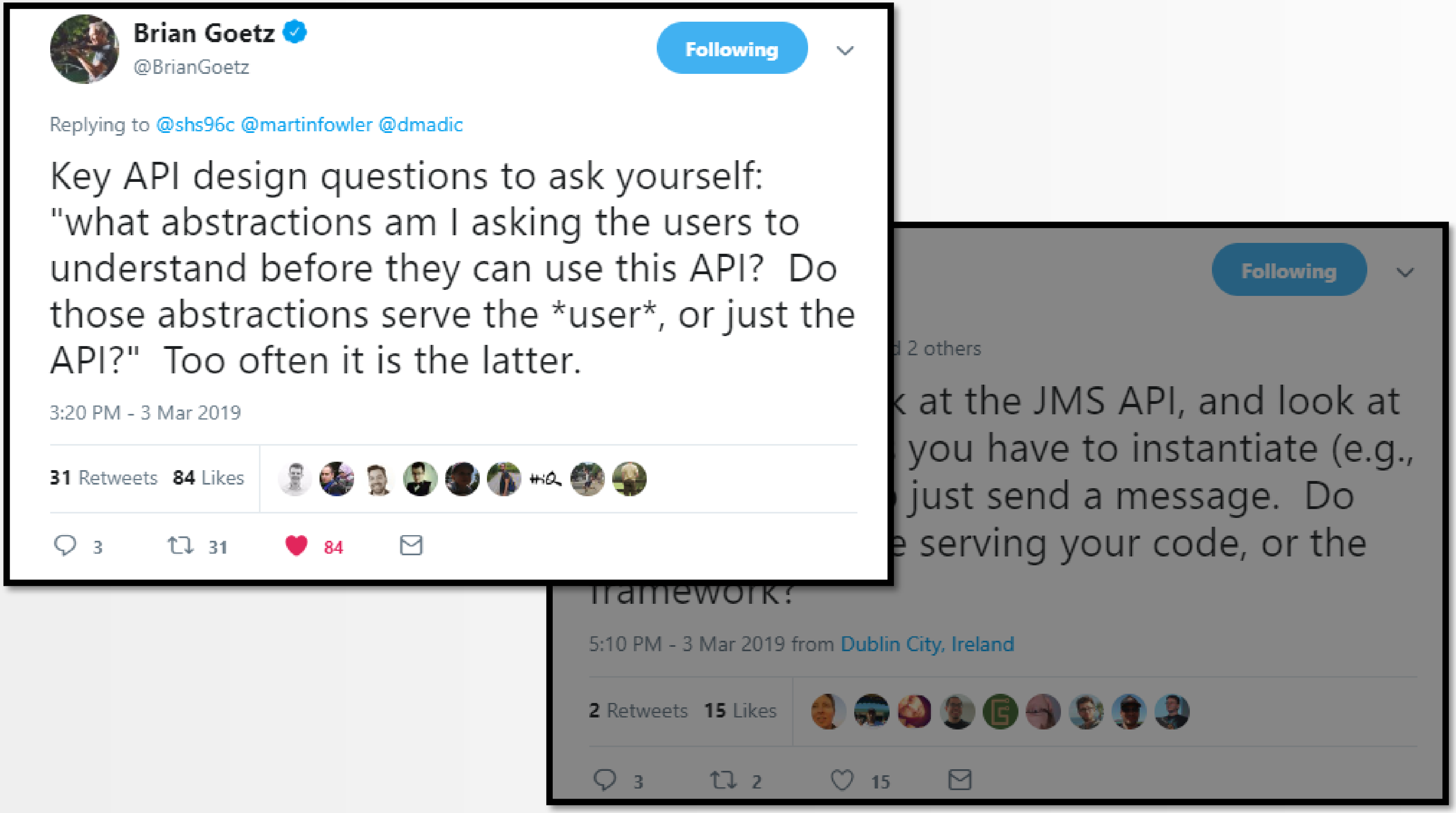
Consistency is at the core of usability

7. Consistency

All of these are affected!

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?
>
> Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
>
> Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
>
> Satisfaction: How pleasant is it to use the design?
>
> https://www.nngroup.com/articles/usability-101-introduction-to-usability/

8. Convenience



Brian Goetz
@BrianGoetz

Following

Replying to @shs96c @martinfowler @dmadic

Key API design questions to ask yourself: "what abstractions am I asking the users to understand before they can use this API? Do those abstractions serve the *user*, or just the API?" Too often it is the latter.

3:20 PM - 3 Mar 2019

31 Retweets 84 Likes

3 31 84

Following

d 2 others

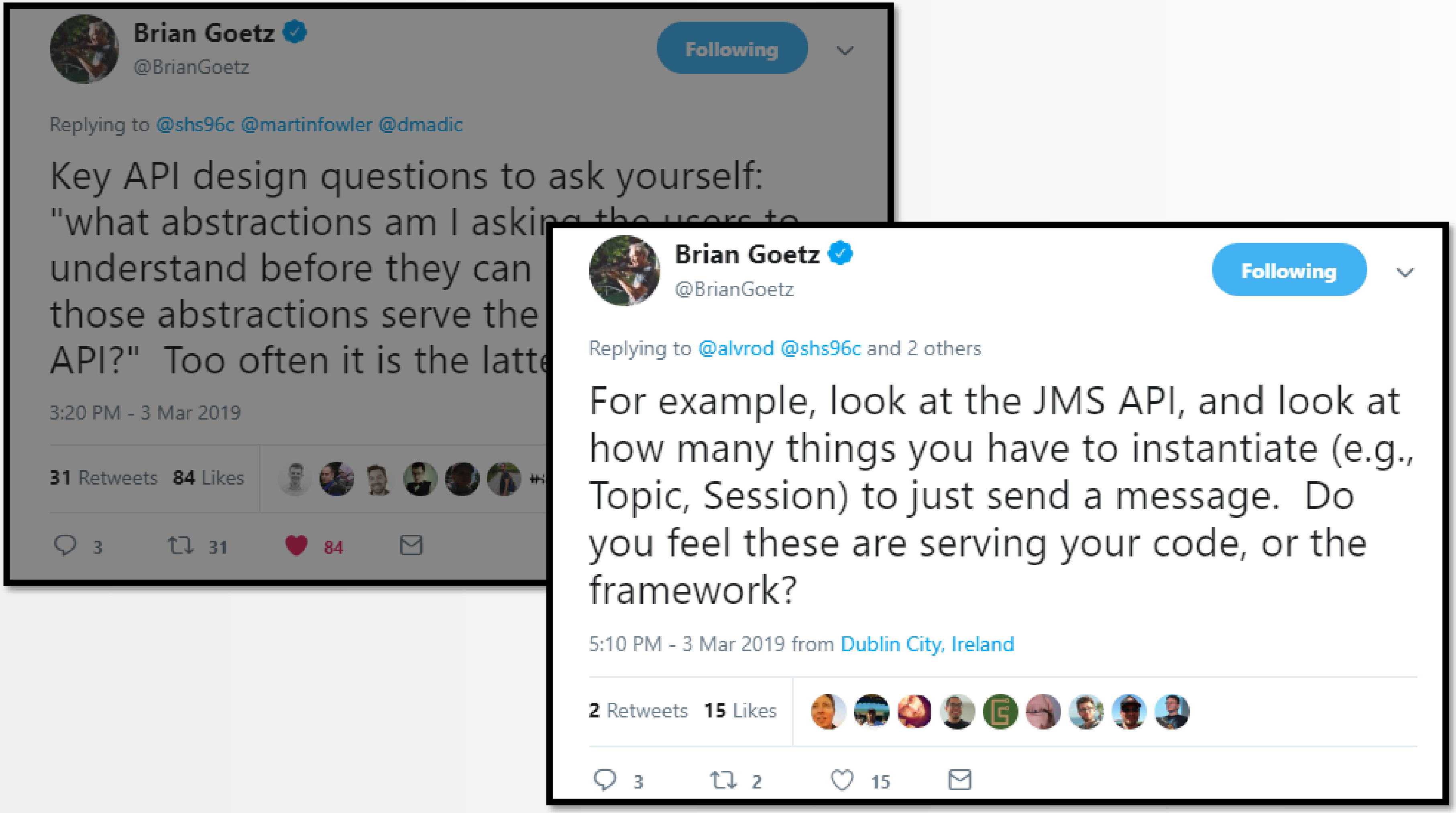
k at the JMS API, and look at
you have to instantiate (e.g.,
just send a message. Do
e serving your code, or the
framework?

5:10 PM - 3 Mar 2019 from Dublin City, Ireland

2 Retweets 15 Likes

3 2 15

8. Convenience



Brian Goetz
@BrianGoetz

Replying to @shs96c @martinfowler @dmadic

Key API design questions to ask yourself: "what abstractions am I asking the users to
understand before they can
those abstractions serve the
API?" Too often it is the latte

3:20 PM - 3 Mar 2019

31 Retweets 84 Likes

Brian Goetz
@BrianGoetz

Replying to @alvrod @shs96c and 2 others

For example, look at the JMS API, and look at how many things you have to instantiate (e.g., Topic, Session) to just send a message. Do you feel these are serving your code, or the framework?

5:10 PM - 3 Mar 2019 from Dublin City, Ireland

2 Retweets 15 Likes

8. Convenience

```java
// Ignoring exceptions...
InputStream is = ...
OutputStream os = ...

byte[] buffer = new byte[1024];
int length;
while ((length = is.read(buffer)) > -1) {
    baos.write(buffer, 0, length);
}
```

8. Convenience

9 (!) major releases later…

8. Convenience

```
public abstract class InputStream {

    ...

    /**
     * @since 9
     */
    public long transferTo(OutputStream out) throws IOException {
        long transferred = 0;
        byte[] buffer = new byte[DEFAULT_BUFFER_SIZE];
        int read;
        while ((read = this.read(buffer, 0, DEFAULT_BUFFER_SIZE)) >= 0) {
            out.write(buffer, 0, read);
            transferred += read;
        }
        return transferred;
    }
}
```

8. Convenience

```
public abstract class

    ...

    /**
     * @since 9
     */
    public long trans                                        on {
        long transfer
        byte[] buffer
        int read;
        while ((read                                         ZE)) >= 0) {
            out.write
            transferr
        }
        return transf
    }
}
```

8. Convenience

Convenience is the most underrated API feature

… and language feature!

8. Convenience – Dogfooding

How to achieve convenience?

… by dogfooding

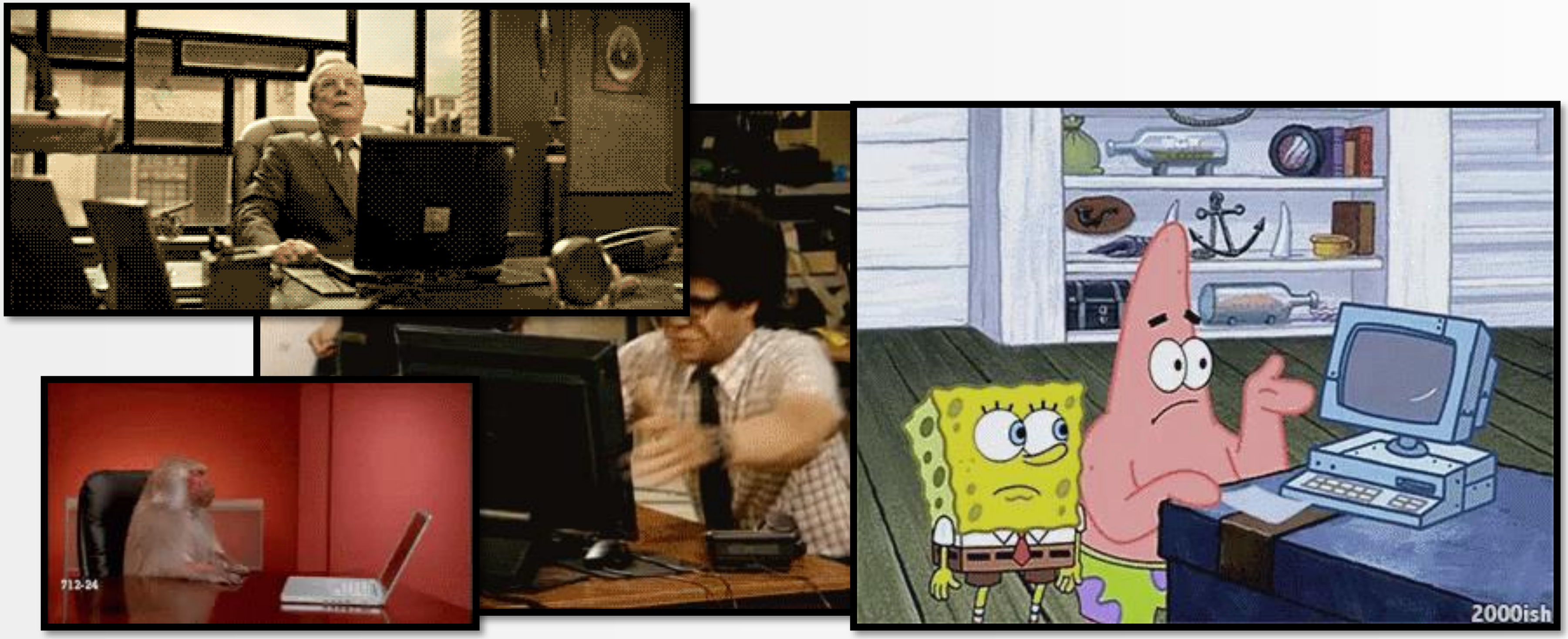
8. Convenience – Dogfooding

Use your own API all the time.

… by eating your own «dog food»

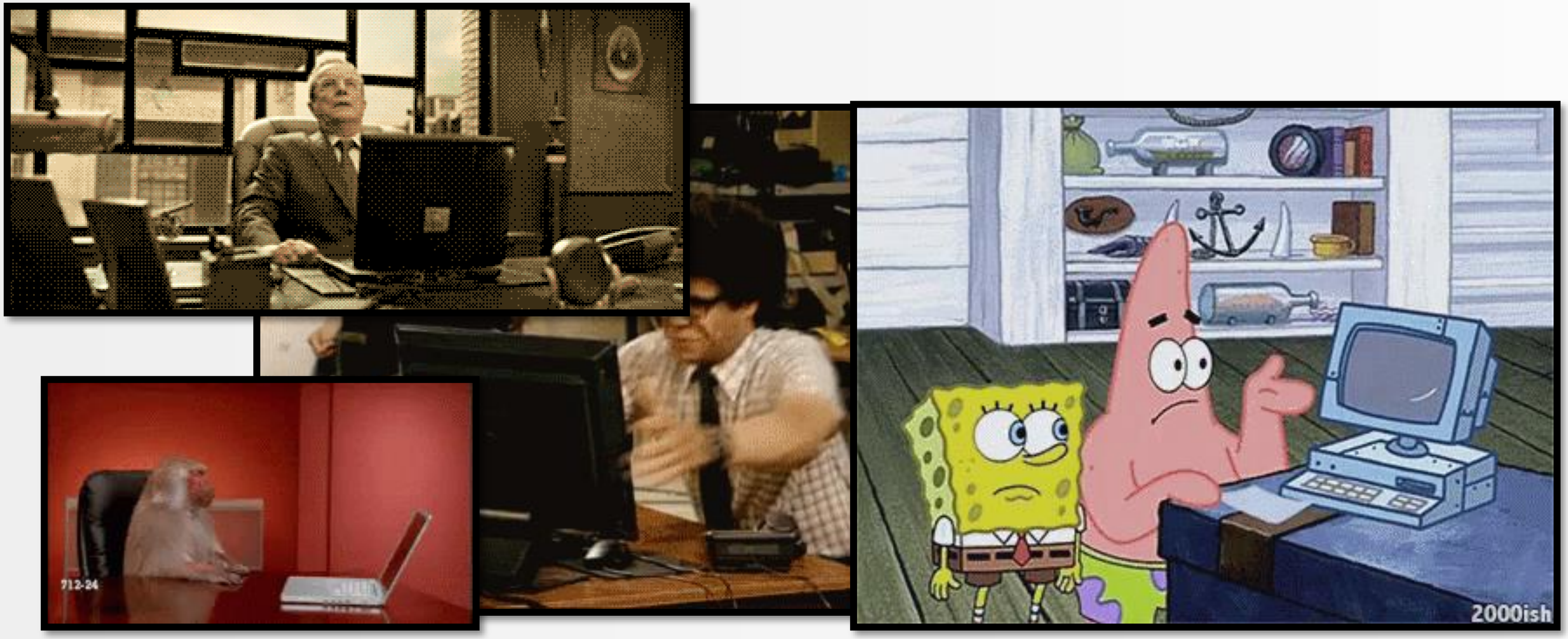
8. Convenience – Dogfooding

Me, when I use my own API



8. Convenience – Dogfooding

Me, when I use my *any* API



8. Convenience – Dogfooding

But at least I can fix my own

8. Convenience – Dogfooding

Advantages:

8. Convenience – Dogfooding

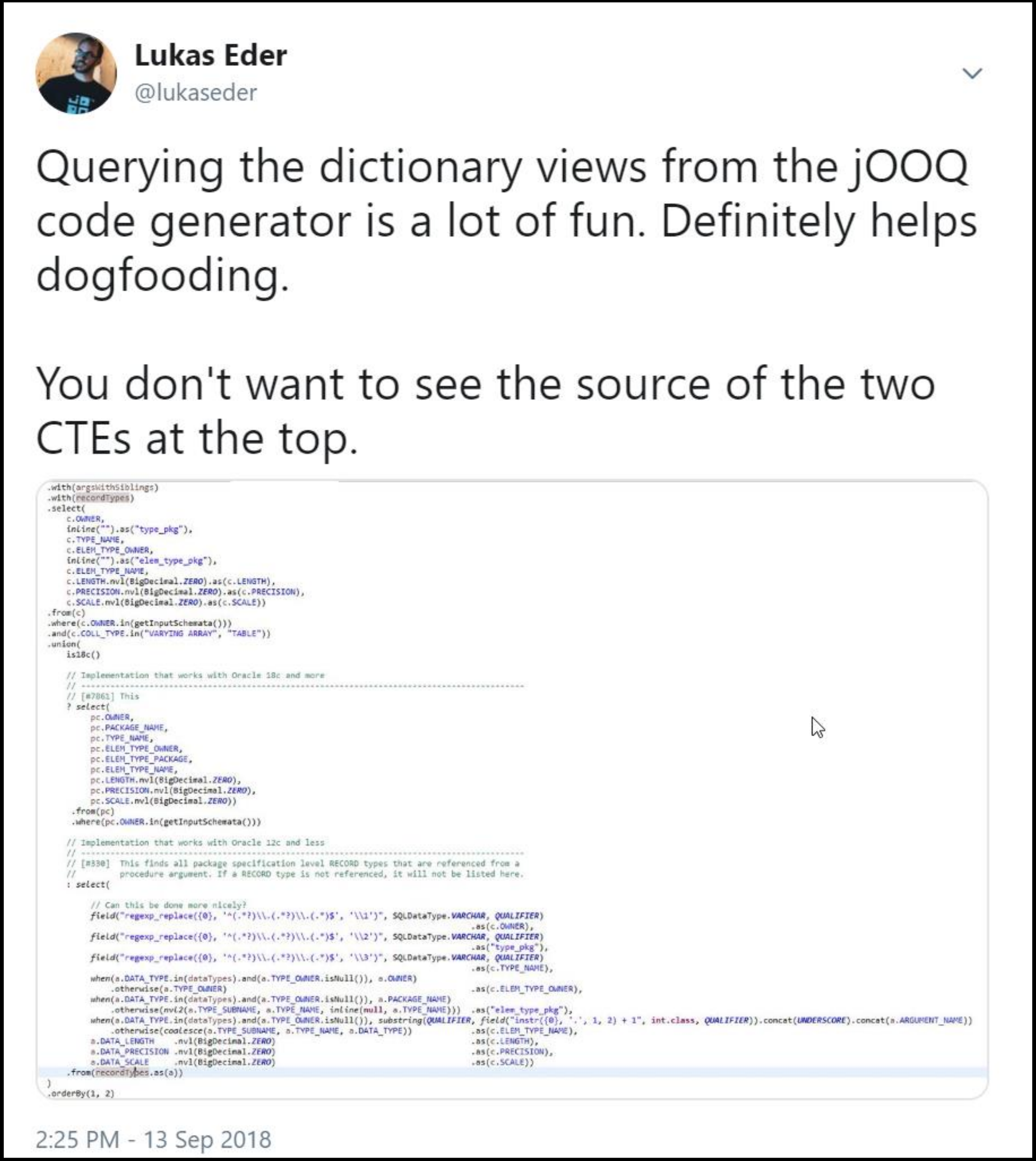
Advantages:

- Better objective quality

8. Conve

Adv

– E



Lukas Eder
@lukaseder

Querying the dictionary views from the jOOQ code generator is a lot of fun. Definitely helps dogfooding.

You don't want to see the source of the two CTEs at the top.

```
.with(argsWithSiblings)
.with(recordTypes)
.select(
    c.OWNER,
    inline("").as("type_pkg"),
    c.TYPE_NAME,
    c.ELEM_TYPE_OWNER,
    inline("").as("elem_type_pkg"),
    c.ELEM_TYPE_NAME,
    c.LENGTH.nvl(BigDecimal.ZERO).as(c.LENGTH),
    c.PRECISION.nvl(BigDecimal.ZERO).as(c.PRECISION),
    c.SCALE.nvl(BigDecimal.ZERO).as(c.SCALE))
.from(c)
.where(c.OWNER.in(getInputSchemata()))
.and(c.COLL_TYPE.in("VARYING ARRAY", "TABLE"))
.union(
    is18c()

    // Implementation that works with Oracle 18c and more
    // -----------------------------------------------------------------------------------
    // [#7861] This
    ? select(
         pc.OWNER,
         pc.PACKAGE_NAME,
         pc.TYPE_NAME,
         pc.ELEM_TYPE_OWNER,
         pc.ELEM_TYPE_PACKAGE,
         pc.ELEM_TYPE_NAME,
         pc.LENGTH.nvl(BigDecimal.ZERO),
         pc.PRECISION.nvl(BigDecimal.ZERO),
         pc.SCALE.nvl(BigDecimal.ZERO))
     .from(pc)
     .where(pc.OWNER.in(getInputSchemata()))

    // Implementation that works with Oracle 12c and less
    // -----------------------------------------------------------------------------------
    // [#330]  This finds all package specification level RECORD types that are referenced from a
    //         procedure argument. If a RECORD type is not referenced, it will not be listed here.
    : select(

         // Can this be done more nicely?
         field("regexp_replace({0}, '^(.*?)\\.(.*?)\\.(.*)$', '\\1')", SQLDataType.VARCHAR, QUALIFIER)
                                                                                  .as(c.OWNER),
         field("regexp_replace({0}, '^(.*?)\\.(.*?)\\.(.*)$', '\\2')", SQLDataType.VARCHAR, QUALIFIER)
                                                                                  .as("type_pkg"),
         field("regexp_replace({0}, '^(.*?)\\.(.*?)\\.(.*)$', '\\3')", SQLDataType.VARCHAR, QUALIFIER)
                                                                                  .as(c.TYPE_NAME),
         when(a.DATA_TYPE.in(dataTypes).and(a.TYPE_OWNER.isNull()), a.OWNER)
             .otherwise(a.TYPE_OWNER)                                             .as(c.ELEM_TYPE_OWNER),
         when(a.DATA_TYPE.in(dataTypes).and(a.TYPE_OWNER.isNull()), a.PACKAGE_NAME)
             .otherwise(nvl2(a.TYPE_SUBNAME, a.TYPE_NAME, inline(null, a.TYPE_NAME)))  .as("elem_type_pkg"),
         when(a.DATA_TYPE.in(dataTypes).and(a.TYPE_OWNER.isNull()), substring(QUALIFIER, field("instr({0}, '.', 1, 2) + 1", int.class, QUALIFIER)).concat(UNDERSCORE).concat(a.ARGUMENT_NAME))
             .otherwise(coalesce(a.TYPE_SUBNAME, a.TYPE_NAME, a.DATA_TYPE))       .as(c.ELEM_TYPE_NAME),
         a.DATA_LENGTH    .nvl(BigDecimal.ZERO)                                   .as(c.LENGTH),
         a.DATA_PRECISION .nvl(BigDecimal.ZERO)                                   .as(c.PRECISION),
         a.DATA_SCALE     .nvl(BigDecimal.ZERO)                                   .as(c.SCALE))
    .from(recordTypes.as(a))
)
.orderBy(1, 2)
```

2:25 PM - 13 Sep 2018

8. Convenience – Dogfooding

Advantages:

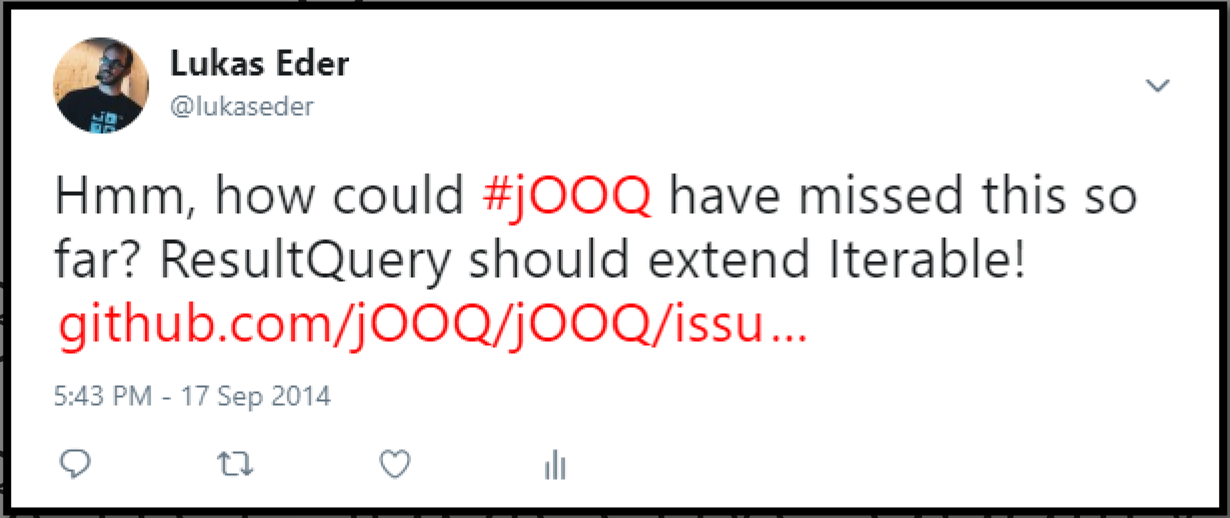
- Better objective quality
- Better subjective quality

8. Convenience – Dogfooding

Advantages:

- B

- Better subjective quality



Lukas Eder
@lukaseder

Hmm, how could #jOOQ have missed this so far? ResultQuery should extend Iterable! github.com/jOOQ/jOOQ/issu…

5:43 PM - 17 Sep 2014

8. Convenience – Dogfooding

```java
// After 1000x doing this
for (var record : ctx.select(ACTOR.FIRST_NAME, ACTOR.LAST_NAME)
                     .from(ACTOR)
                     .orderBy(ACTOR.ID)
                     .fetch()) {

    ...
}
```

8. Convenience – Dogfeeding

This gets on my nerves!

```
// After 1000x doing this
for (var record : ctx.select(ACTOR.FIRST_NAME, ACTOR.LAST_NAME)
                     .from(ACTOR)
                     .orderBy(ACTOR.ID)
                     .fetch()) {

    ...
}
```

8. Convenience – Dogfooding

```
// ... why not just do it like this?
for (var record : ctx.select(ACTOR.FIRST_NAME, ACTOR.LAST_NAME)
                     .from(ACTOR)
                     .orderBy(ACTOR.ID)) {


    ...
}
```

8. Convenience – Dogfeeding

ResultQuery<R> extends Iterable<R>

```
// ... why not just do it like this?
for (var record : ctx.select(ACTOR.FIRST_NAME, ACTOR.LAST_NAME)
                     .from(ACTOR)
                     .orderBy(ACTOR.ID)) {


    ...
}
```

8. Convenience – Dogfooding

```
-- If you’re coding PL/SQL, this is natural
for record in        (select ACTOR.FIRST_NAME, ACTOR.LAST_NAME
                      from ACTOR
                      order by ACTOR.ID) loop

    ...
end loop
```

8. Convenience

Convenience is not a game changer

8. Convenience

Convenience does not solve «the big problems»

8. Convenience

But convenience makes people happy

8. Convenience

Happy people will recommend your API

8. Convenience – In the Java language

```
// Java 7
try (Statement s = connection.createStatement()) {
  ..
}
```

8. Convenience – In the Java language

```
// Java 7
try (Statement s = connection.createStatement()) {
  ..
}

// Java 6 (more or less)
Statement s = null;
try {
  s = connection.createStatement();
  ..
}
finally {
  if (s != null) try {
    s.close();
  }
  catch (SQLException ignore) {} // Will never happen ;-)
}
```

9. Compatibility

Or in short:

You can only lose

9. Compatibility – Don’t be like Python

```
# Python 2
# --------


> 5/2
2
```

```
# Python 3
# --------


> 5/2
2.5
```

9. Compatibility – Don’t be like Python

```
# Python 2
# --------

> print ("hi")
hi

> print "hi"
hi
```

```
# Python 3
# --------

> print ("hi")
hi

> print "hi"
SyntaxError
```

9. Compatibility – Don’t be like Python

Little is gained from such incompatible changes

9. Compatibility – The other extreme

```
// Java
// ----------

package java.lang;

public final class Boolean {
  public static boolean getBoolean(String name) {
    xxxxxxx xxxxxx = xxxxx;
    xxx {
      xxxxxx = xxxxxxxxxxxx(xxxxxx.xxxxxxxxxxx(xxxx));
    } xxxxx (...) {
    }
    xxxxx xxxxxx;
  }
}
```

9. Compatibility – The other extreme

```
// Java
// ----------

package java.lang;

public final class Boolean {
  public static boolean getBoolean(String name) {
    boolean result = false;
    try {
      result = parseBoolean(System.getProperty(name)
    } catch (...) {
    }
    return result;
  }
}
```

9. Compatibility – The other extreme

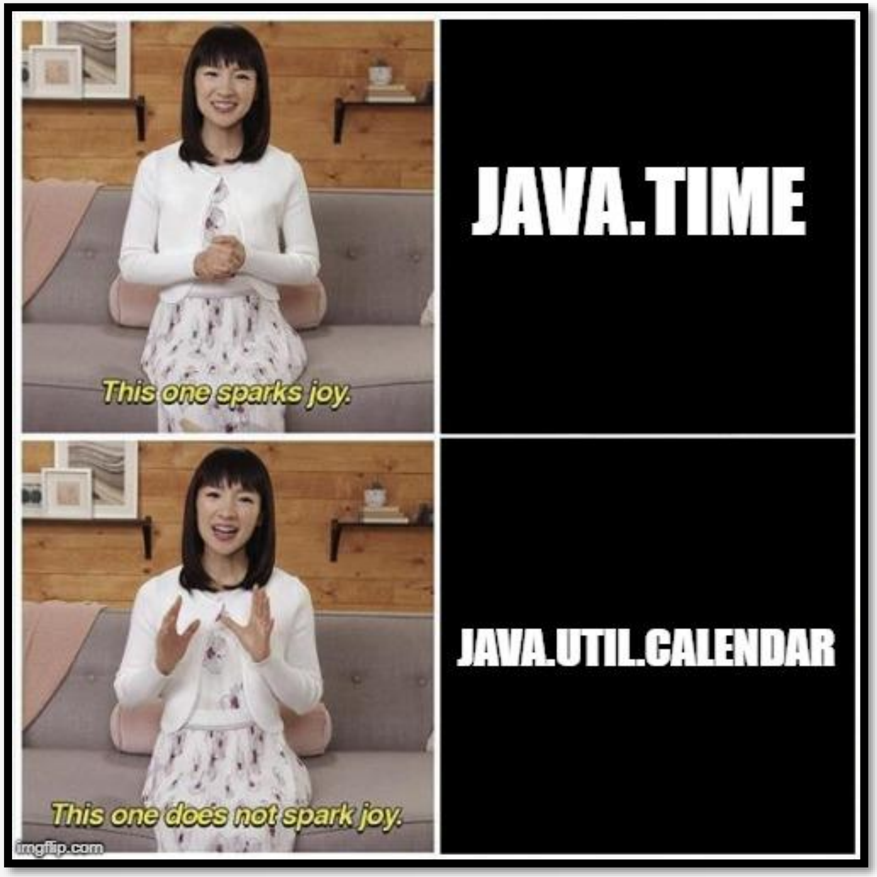
These things are bad because of their lack of consistency

9. Compatibility – The other extreme

Why are they not deprecated and removed?

9. Compatibility – The other extreme

We need more Marie Kondo in the JDK

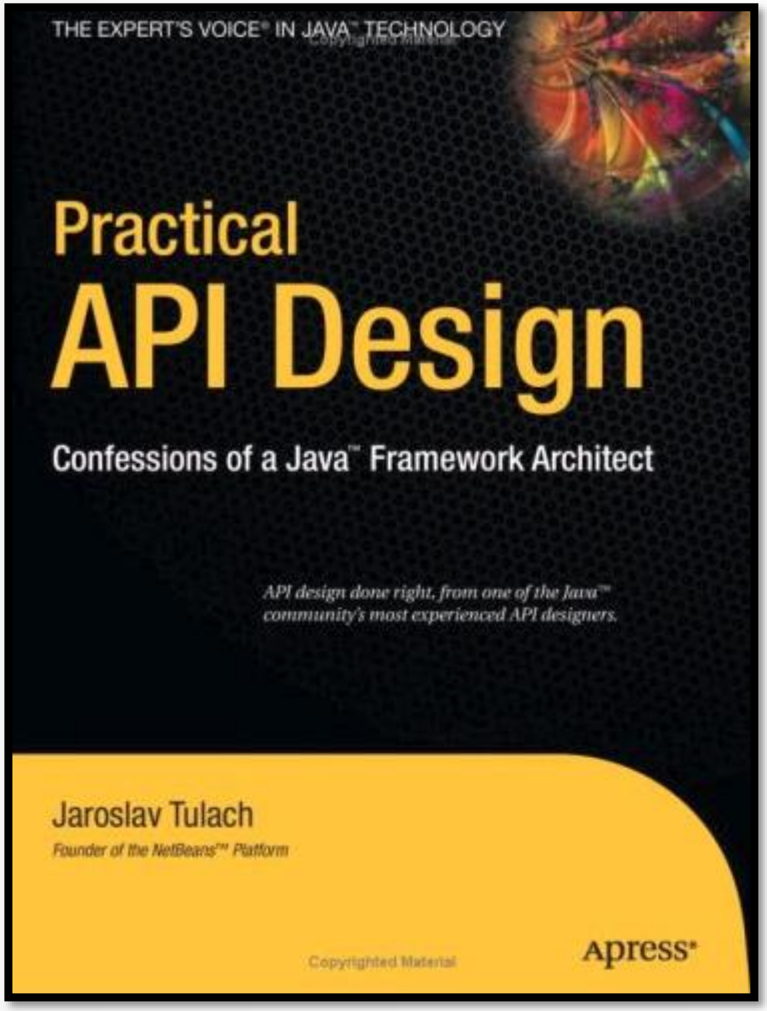


9. Compatibility – Be pragmatic

Be pragmatic

9. Compatibility – A Trick

Keep old API tests around!


THE EXPERT'S VOICE® IN JAVA™ TECHNOLOGY
Practical
API Design
Confessions of a Java™ Framework Architect
API design done right, from one of the Java™ community's most experienced API designers.
Jaroslav Tulach
Founder of the NetBeans™ Platform
Apress®

http://wiki.apidesign.org/wiki/Never_update_tests

9. Compatibility – A Collection of Tricks

ECLIPSE FOUNDATION

https://wiki.eclipse.org/Evolving_Java-based_APIs

9. Compatibility – A Collection of Tricks

https://wiki.eclipse.org/Evolving_Java-based_APIs

https://wiki.eclipse.org/Evolving_Java-based_APIs_2

9. Compatibility – A Collection of Tricks

https://wiki.eclipse.org/Evolving_Java-based_APIs

https://wiki.eclipse.org/Evolving_Java-based_APIs_2

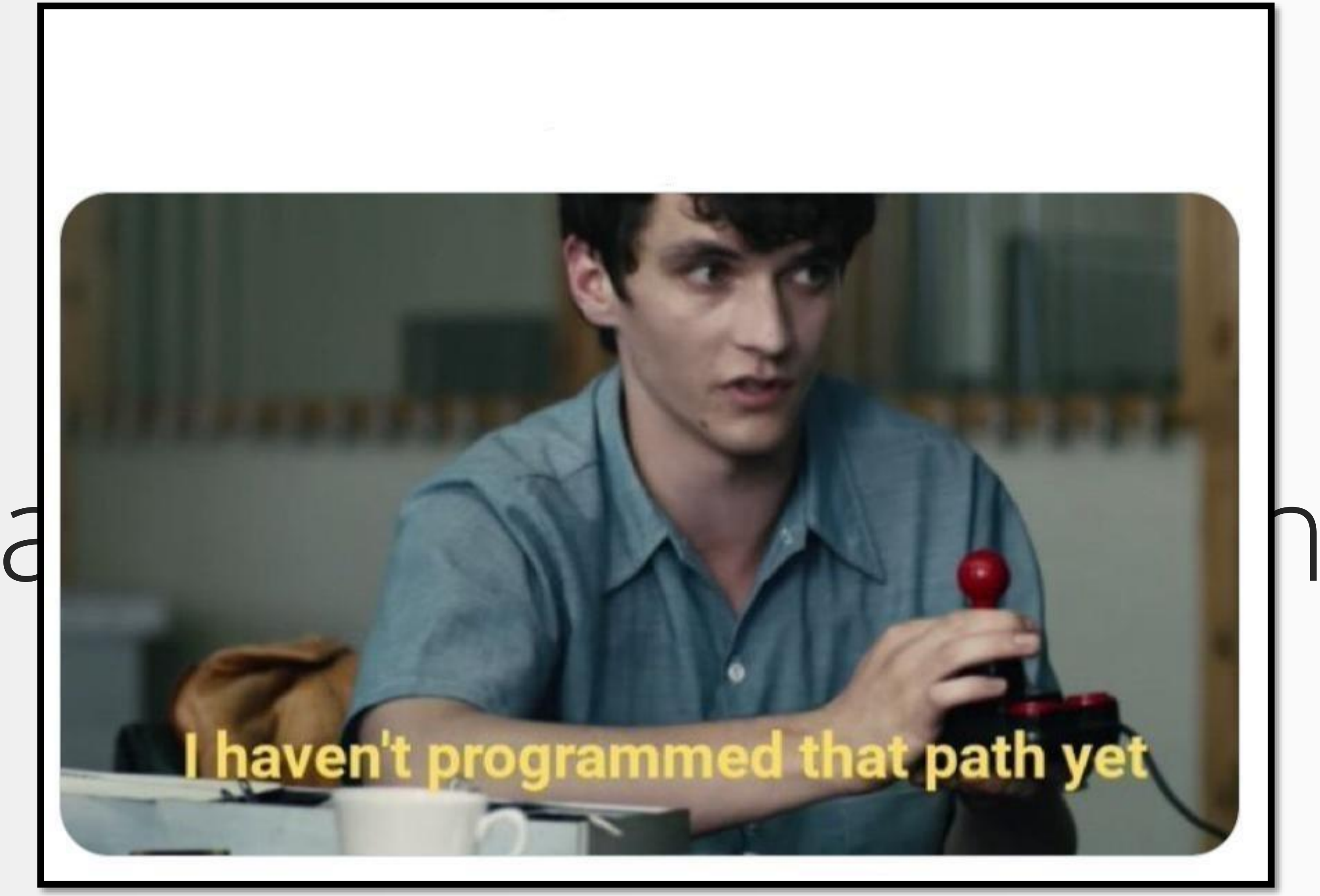
https://wiki.eclipse.org/Evolving_Java-based_APIs_3

Evolving URLs, too

10. Documentation

How developers feel about documentation

10. Documentation

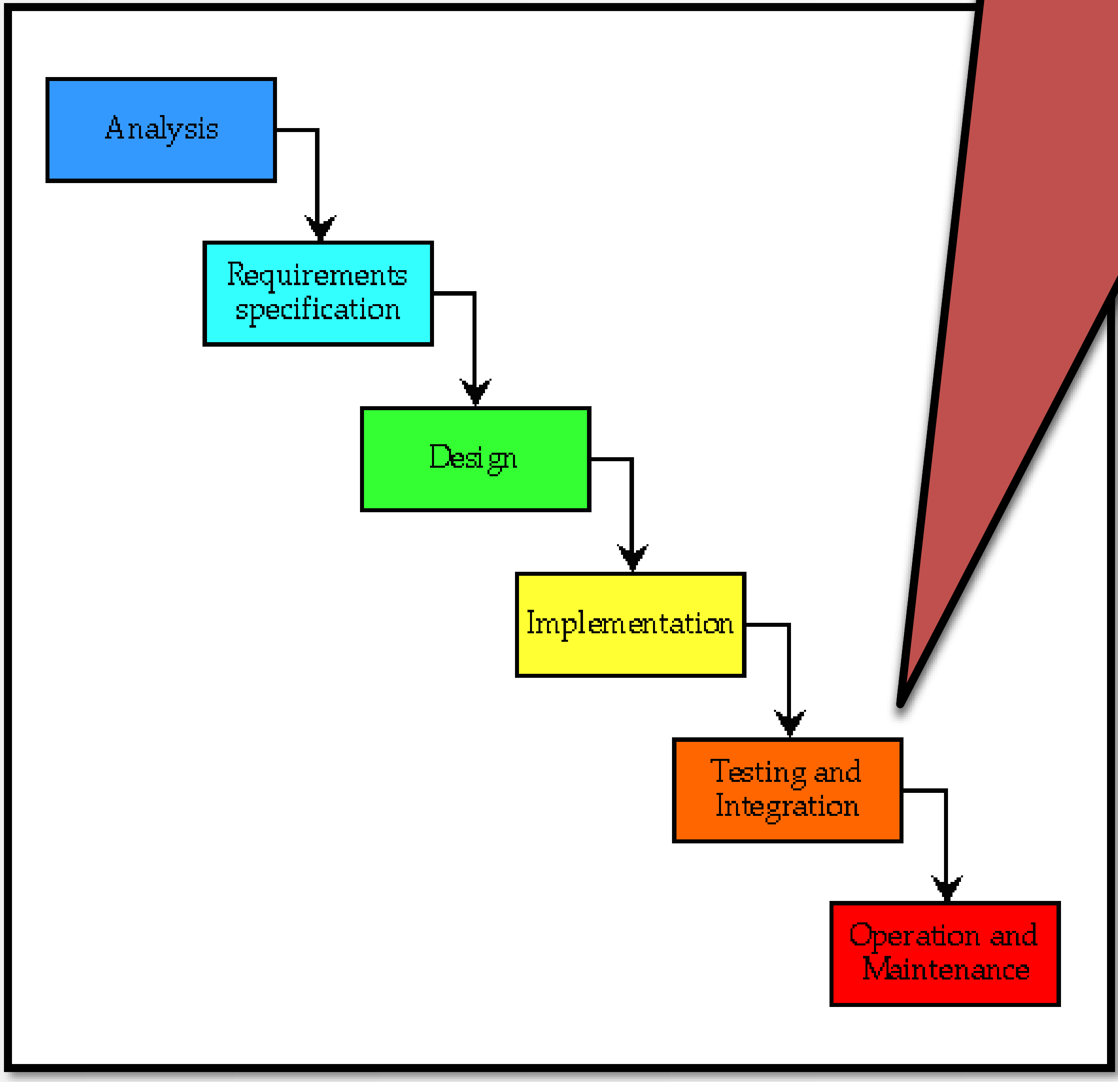


10. Documentation

Just like testing

10. Documentati

We do that at the end, maybe



10. Documentation

```java
public class Customer {





  public String getFirstName() { ... }
}
```

10. Documentatic

Architect: «Document Your Code!!»

```
public class Custo




  public String getFirstName() { ... }
}
```

10. Documentation

```
public class Customer {

  /**
   * Gets the first name.
   *
   * @return the first name.
   */
  public String getFirstName() { ... }
}
```

10. Documentation

Documentation is the dual of discoverability

10. Documentation

Discoverability:

I don’t know what I need.
API, what do you have to offer?

10. Documentation

Documentation:

I need this thing.
API, do you happen to support it?

Overview

10 Reasons Why we Love Some APIs and Why we Hate Some Others

1. Naming
2. Simplicity
3. Do One Thing
4. Types
5. Discoverability
6. Error Handling
7. Consistency
8. Convenience
9. Compatibility
10. Documentation

Overview

10 Reasons Why we Love Some APIs and Why we Hate Some Others

1. Naming
2. Simplicity
3. Do One Thing
4. Types
5. Discoverability
6. Error Handling
7. Consistency
8. Convenience
9. Compatibility
10. Documentation

Remember: NSDTDECCCD

D is silent

Overview

Design for humans

Programmers are humans too

Overview

Want proof?

HOW TO INSULT A DEVELOPER
geek & poke
IT'S NOT RESTFUL

Geek and Poke. Licensed CC-BY-3.0.
http://geek-and-poke.com/geekandpoke/2013/6/14/insulting-made-easy

Overview

> Usability is defined by **5 quality components**:
>
> Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
>
> Efficiency: Once users have learned the design, how quickly can they perform tasks?
>
> Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
>
> Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
>
> Satisfaction: How pleasant is it to use the design?

https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Thank you

Check out our trainings:

http://www.jooq.org/training

Coordinates

- Blog: http://blog.jooq.org (excellent Java SQL content)
- Twitter: @JavaOOQ / @lukaseder (more lame jokes)
- E-Mail: lukas.eder@datageekery.com
- Bank account: CH57 8148 7000 0SQL AWSM 7