



The Proxy Fairy

The Magic of Spring

When something is **painful**
but you can't avoid doing it...

postpone it


When something is **painful**
but you can't avoid doing it...

delegate it

When something is **painful**
but you can't avoid doing it...

Do It More Often!

"Bring The Pain Forward!"

A lion with a large, golden-brown mane is roaring with its mouth wide open, showing its teeth and tongue. The lion is standing on a rocky outcrop. The background is a savanna landscape under a dramatic, golden sunset sky with scattered clouds. The overall tone is warm and intense.

XIP

"Bring The Pain Forward!"

Continuous Integration

Pair Programming

Continuous Refactoring

TDD



Victor Rentea

14 years of Java

Scala
.NET
PHP

Lead Architect

Tech Team Lead and Consultant



Software Craftsman

XP: Pair Programming, Refactoring, TDD

Clean Code Evangelist

30+ talks, 12 meetups

best here:

VictorRentea.ro

Independent

Technical Trainer & Coach

6 years

1300 devs

180+ days

30 companies

Spring



Hibernate



Java 8



Design Patterns



Architecture, DDD

Clean Code

Unit Testing, TDD

Java Performance

Scala

and much more...

victor.rentea@gmail.com

VictorRentea.ro

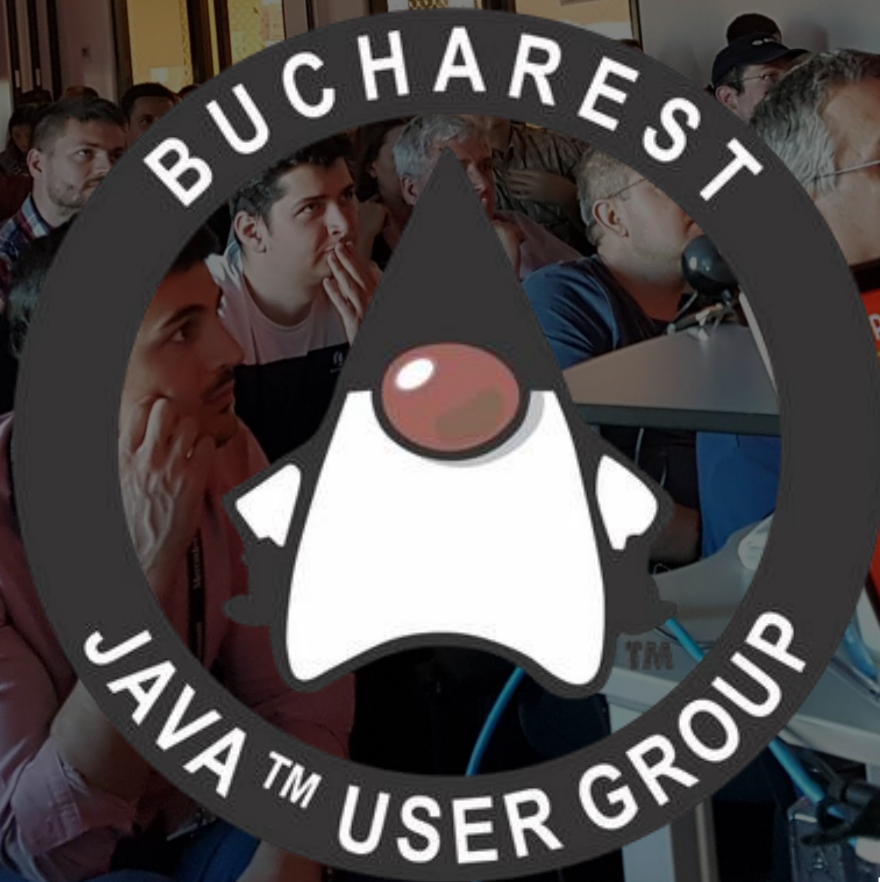
Posting daily on





Lots of people

Largest 3 Bucharest JUG Meetups




Next one: April 17th 2019

Founder of



**Bucharest Software
Craftsmanship Community**

<http://victorrentea.ro/community>

Free Remote Events in  :
#TechyReaders: 1 chapter/week
#LiveCoding Webinars



Passion
Clean Code
Pair Programming
Test-Driven Development
Extreme Programming
Simple Design
Refactoring
Passion

Bucharest Software
Craftsmanship Community

<http://victorrentea.ro/community>

Join us on
meetup



A Success Story



- Log the arguments passed to `m1()` You add a `log.debug(...)` in `m1`
- The same for `m2()`, please You **copy-paste** in `m2` and adjust it
- And for `m3,m4,...m30` ?!!...Argh!!... **copy-paste** ...
- I love it! I want the same Whaat?!!! ... **copy-paste** in 110 methods.
for **all methods** in this package You feel bad. For 5 minutes.
Time passes by ~~~~~ Next task...



A Success Story



Time passes by (2 weeks later)

■ **BUG: a method is not logged !!!%@^!**

I'll put this method here...

■ Please change the log format

Oh boy! 1, 2, 3, .. 109, Done!

■ **BUG: you missed a spot !!!%@^!**

I ♥ my job!





A Success Story



- Log the arguments passed to `m1()` You add a `log.debug(...)` in `m1`
- The same for `m2()`, please You **copy-paste** in `m2` and adjust it
- And for `m3,m4,...m30` ?!!...Argh!!... **copy-paste** ...
- I love it! I want the same Whaat?!!! ... **copy-paste** in 110 methods.
for **all methods** in this package You feel bad. For 5 minutes.

Next task...

~~~~~ Time passes by ~~~~~

# Never **copy-paste** logic

The Capital Sin in Programming

**DRY**  
DON'T REPEAT YOURSELF

decompose it in smaller methods  
and call them as you need



“But I will still need to add one line in each method!”

What if I could magically  
intercept method calls  
and do stuff for each call ?



“Cross-cutting concerns”:

Logging  
Transactions  
Access Control  
Audit

# Aspect-Oriented Programming<sup>...</sup>

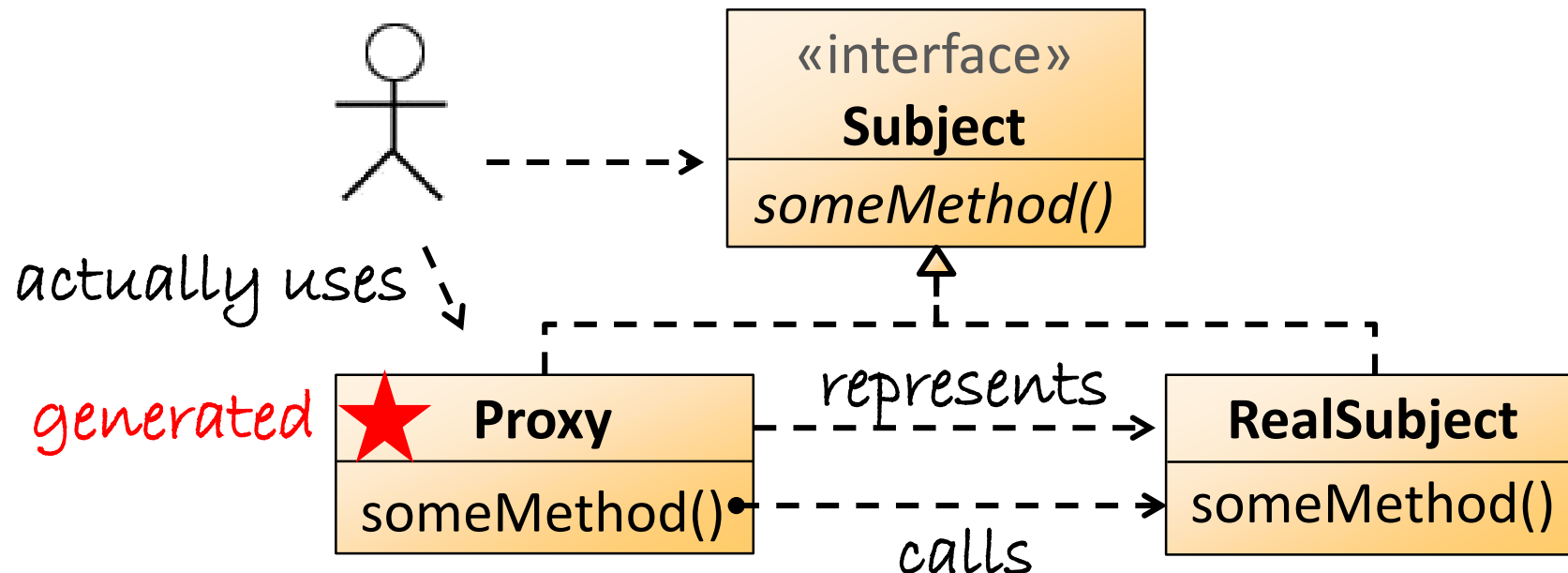
# Proxy

*placeholder intermediating interactions with an object*

- The client wants to work with the Real Subject
- But we trick him to intercept his calls, to do

(1) **AOP**

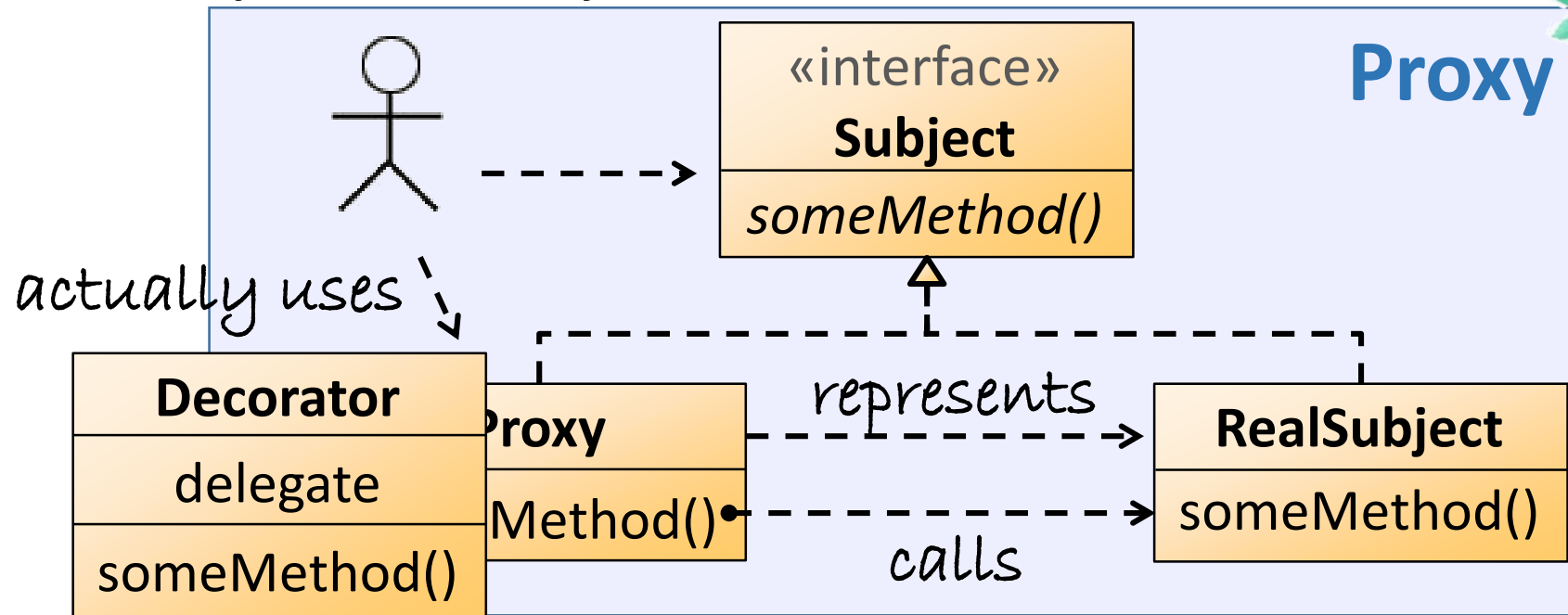
(2) Remote calls (RMI, Web Services)



# Decorator

*do more things in an object's methods, by composition*

- Implement the Subject interface, execute code before/after/instead calling the real method
  - Can be nested: `new Decor1(new Decor2(original));`
  - Usually written by hand



# Contents

- Decorator + Proxy Patterns
- **Interface Proxies**
- **Class Proxies**
- **In-depth Spring AOP**
- Custom Aspects - Best Practices



**KEEP  
CALM  
AND  
CODE**

# The Magic of Spring

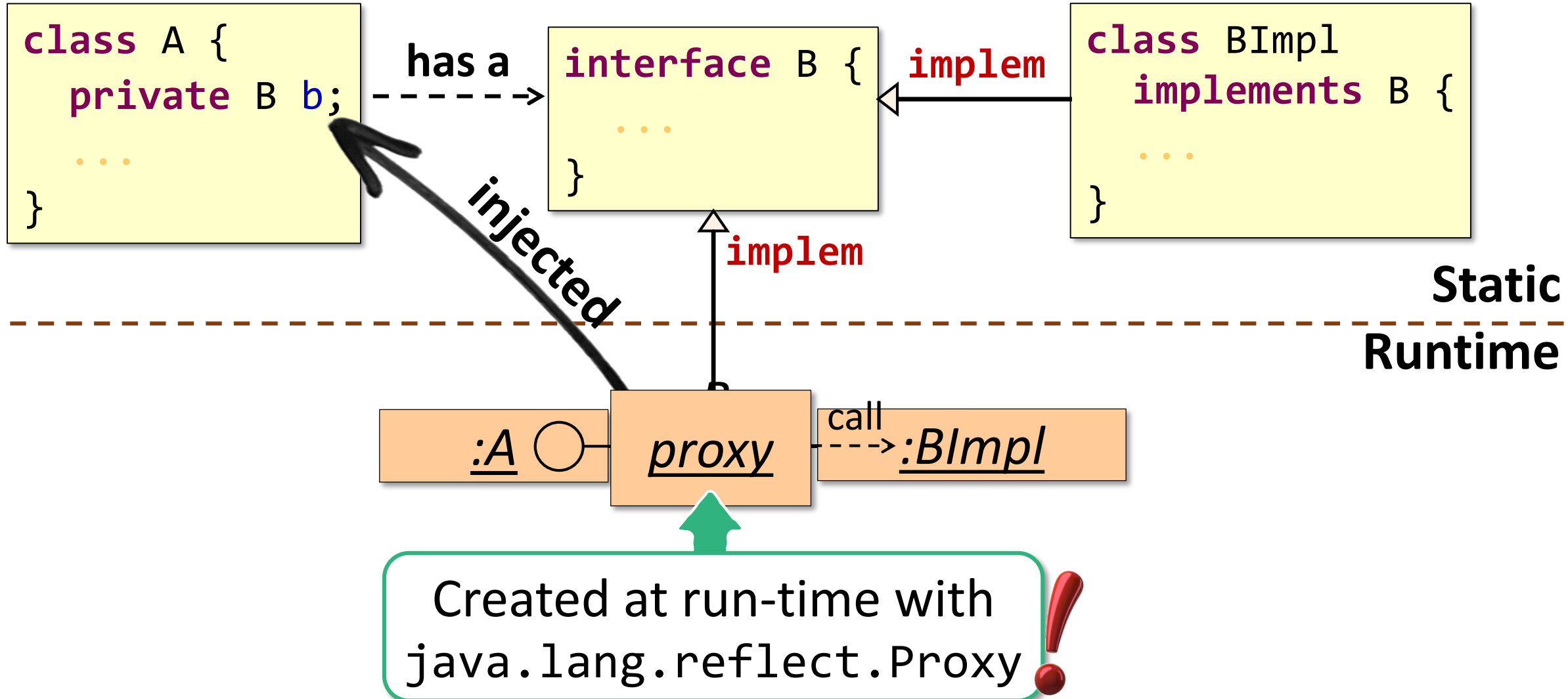
Every time you don't understand how Spring does something...  
**it's a Proxy**

**@Cacheable**  
**@Transactional**  
**@Aspect**  
**@Async**  
**@Secured**  
**@Validated**  
**Request/Thread scope**

...

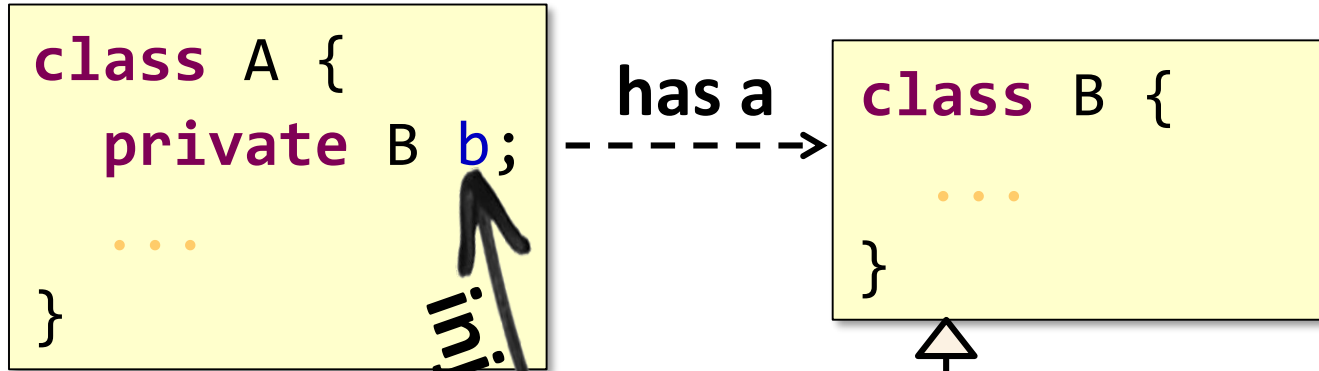


# Interface Proxy



# Class Proxy

identity of the 'b' instance is irrelevant (think 'request' scope)



injected

extends

Static  
Runtime



Concrete classes are proxied via bytecode generation !



# Can AOP Intercept...

• A method of a non-Spring bean? **NO!**

• A final method/class?

• For a class-proxy: **NO!**

• Field access? **NO!**

• A private method call? **NO!**

Well...

Actually...

You know...

You can do any of that via:

**Bytecode Enhancement**

(compile-time hacking)

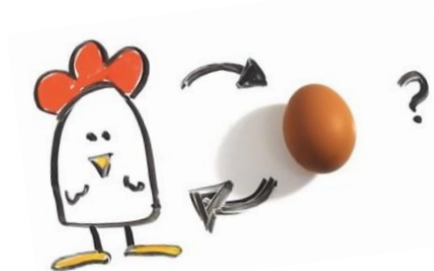
**Or Instrumentation**

(classload-time hacking)

To intercept a local call,

# Call Yourself via a Proxy

- ~~Autowiring yourself ?~~



- `@Autowired ApplicationContext + getBean(T)`
- `@Autowired ObjectFactory<> + getObject()`
- `@Lookup` method 🤪
- `AopContext.currentProxy()` ✓

proxy

# AopContext.currentProxy()

```
@Transactional(propagation = REQUIRES_NEW)  
public void methodWithTx() {
```

```
methodWithTx();  
CurrentClass myselfProxied =  
    (CurrentClass)AopContext.currentProxy();  
myselfProxied.methodWithTx();
```

# No More Magic



- an AOP Alternative -

# Functional Programming

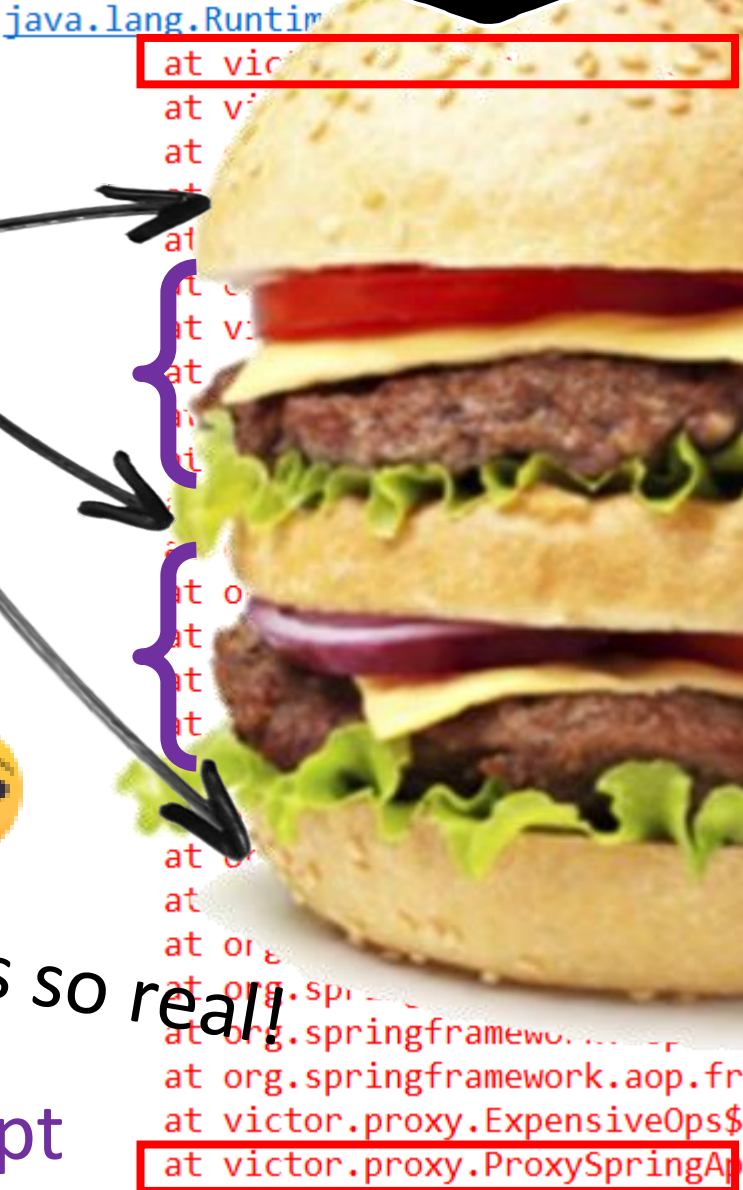
## "Execute Around" Pattern

```
measure(() -> savePage(recordsPage));
```

```
transactionTemplate.execute(this::methodInNewTx);
```

# The Hamburger Problem

stacktrace



I found my code!

What the heck is **the rest**?!

Are you sure you want to know? 😊

**PROXIES**

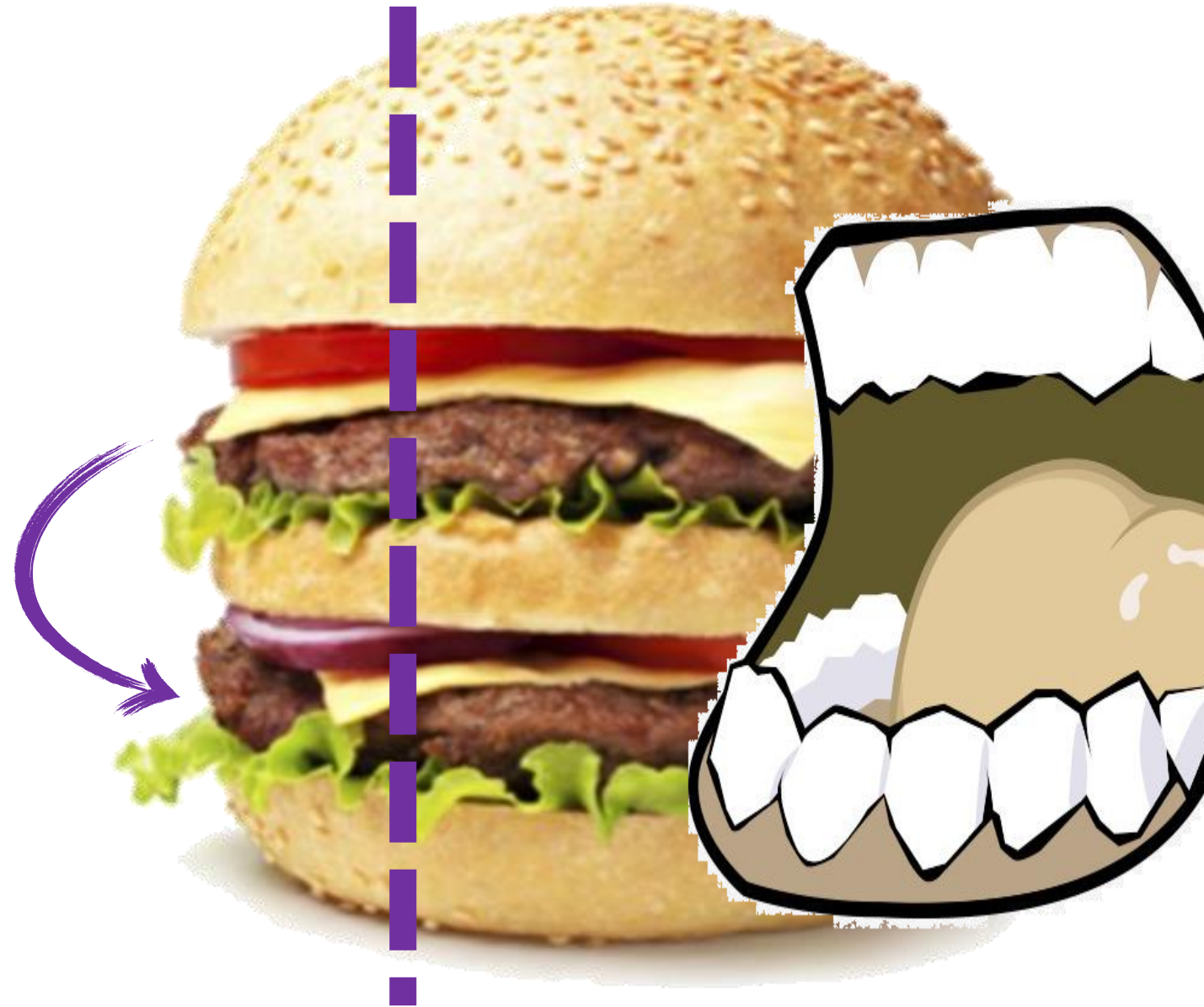
...but it looks so real!

in front of any class with 1+ methods to intercept

**How can Aspects**  
*(invisible logic)*  
**communicate?**

**What is shared?**  
**(within an invocation)**

**ThreadLocal**  
*(invisible data)*







# Propagate Thread Data to worker threads

Hunt me  
afterwards



**ThreadLocal Data** -- what about `@Async` calls?

# Contents

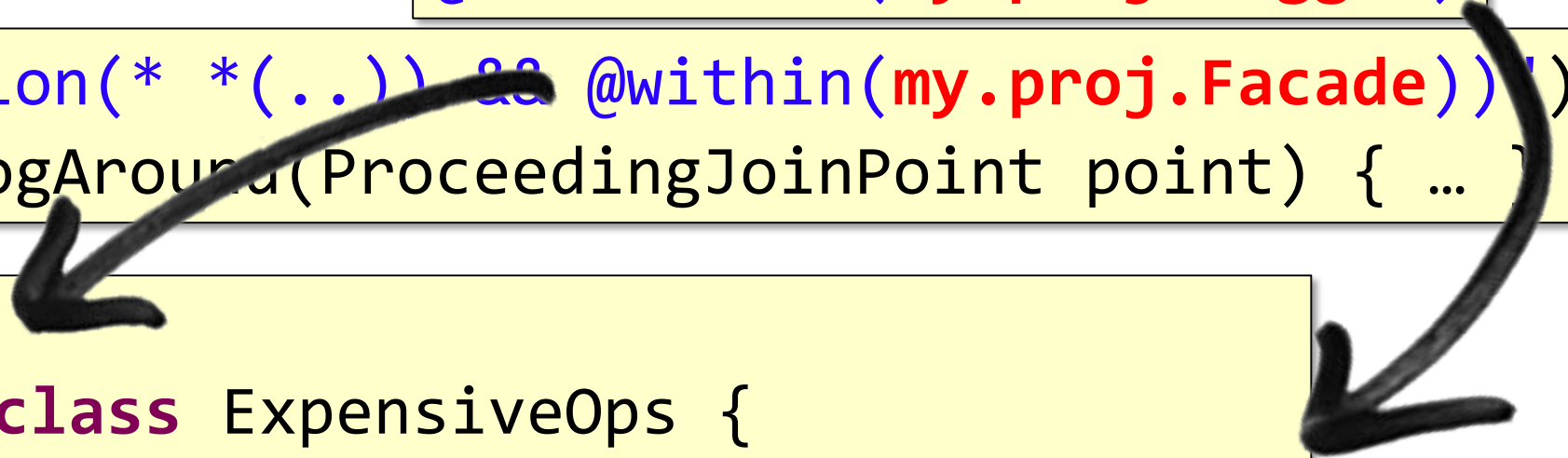
- Decorator + Proxy Patterns
- Interface Proxies
- Class Proxies
- In-depth Spring AOP
- **Custom Aspects - Best Practices**

## annotation-based weaving

**make them visible**

```
@annotation(my.proj.Logged)  
  
@Around("execution(* *(..)) @within(my.proj.Facade)")  
public Object logAround(ProceedingJoinPoint point) { ... }
```

```
@Facade  
public class ExpensiveOps {  
    @Logged  
    public Boolean isPrime(int n) {...}  
}
```



**annotation-based weaving** ✓

instead of

**package name**

```
"execution(* com.mycomp.proj.facade.*.*(..))"
```

or

**class name**

```
"execution(* com.mycomp.*DAO.*(..))"
```

## keep them light

~~synchronized~~

~~files~~

~~database~~

~~http~~

## don't intercept insane-rate calls



## don't intercept insane-rate calls

**I did that!**



alternatives:

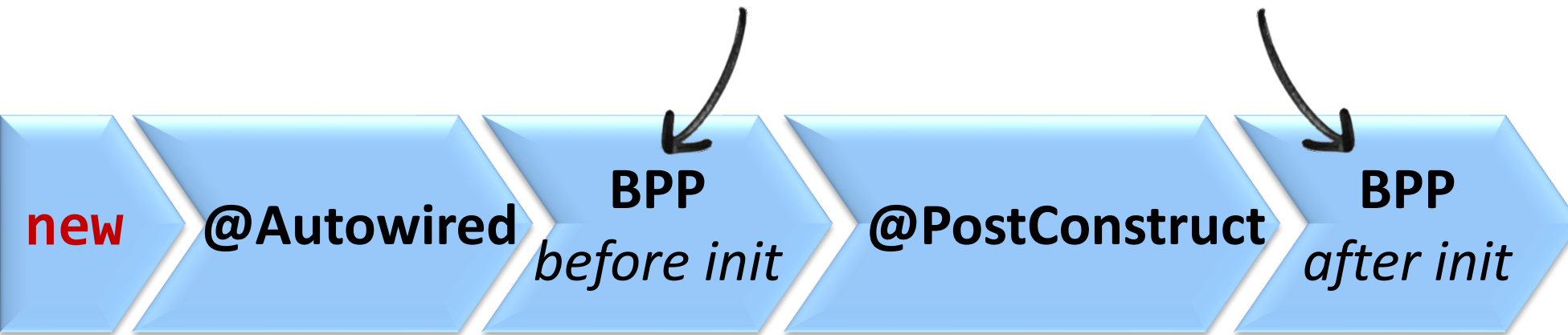
**CPP**

**Execute-Around<sup>FP</sup>**

**BeanPostProcessor**

## BeanPostProcessor

Can return a **proxy** instance



**WHY?**



# Performance

|                   | Average | Error (99.9%) | Unit  |
|-------------------|---------|---------------|-------|
| Cache Method      | 357.7   | ± 11.7        | us/op |
| Decorator         | 355.1   | ± 6.0         |       |
| Interface Proxy   | 398.7   | ± 28.0        |       |
| Class Proxy       | 386.9   | ± 17.5        |       |
| Class Proxy / BPP | 373.9   | ± 36.0        |       |
| @Cacheable        | 8109.8  | ± 1097.5      |       |

Details? Run it yourself: <https://github.com/victorrentea/proxy-fairy-performance>



## Leftover Tricks

Hunt me  
afterwards

- Double proxying
- `@Aspect @Order`
- Sharing the PersistenceContext
- Proxies in ORMs

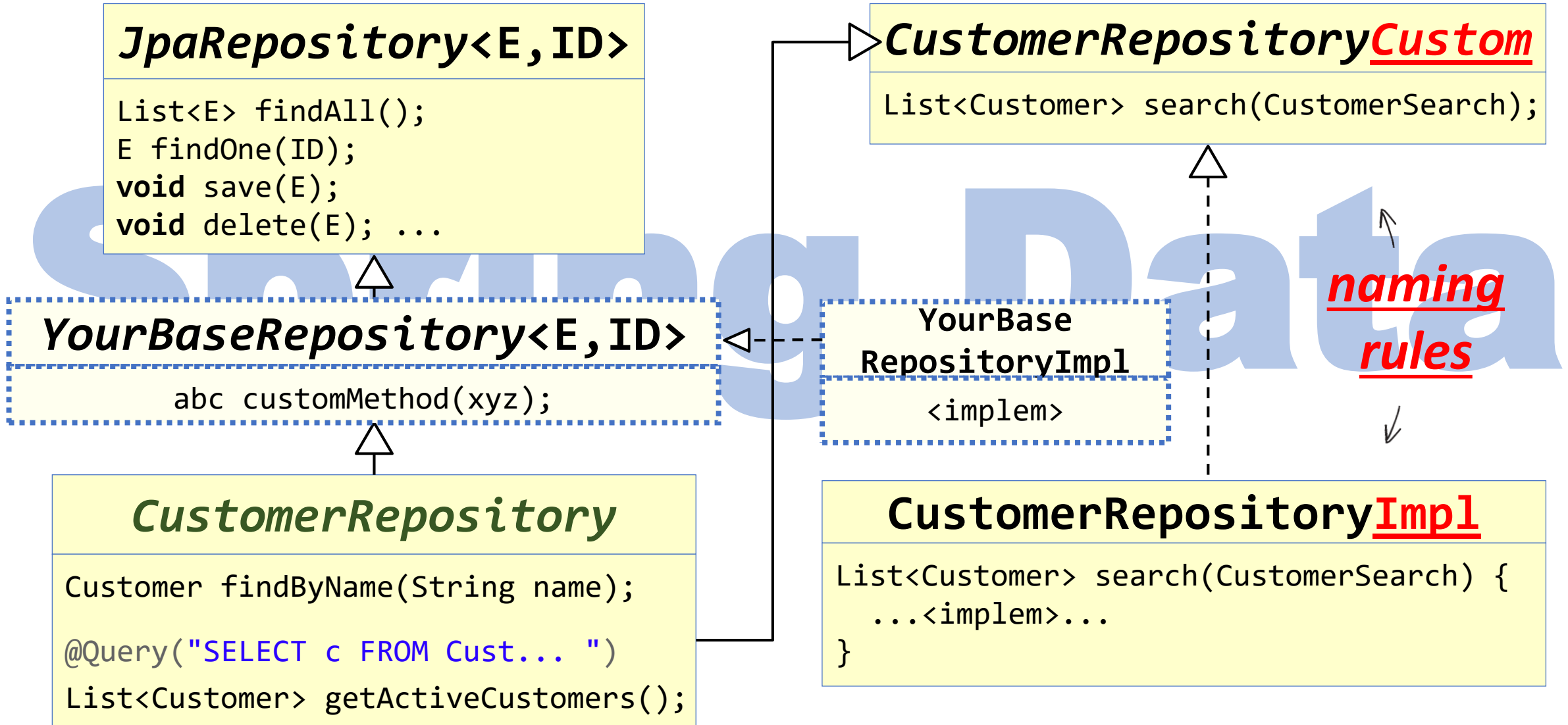
# The Magic of Spring

Every time you don't understand how Spring does something...

**it's a Proxy**

**mixins**

# mixins



# Key Points

- ✓ Proxy + Decorator Patterns; wiring
- ✓ Interface & Class Proxy - under the hood
- ✓ Spring AOP - tricks & limitations
- ✓ Custom Aspects - Best Practices



**Stay on mainstream**

**KISS**

**Understand what's under the hood!  
Know when to get your hands dirty**

# Thank You!

Thanks for review,  
Vladimir Sitnikov



And have a nice Spring!



COME TAKE  
SOME STICKERS:

Stay into  
*The Light*



Functional Party  
*Activist*



I'm available  
*a proof of seniority*



Purpose of code:--Uncle Bob

1. *Maintainable*
2. *Does its job!*

when in doubt  
refactor



*I use both hemispheres*

**Clean Code**

*needs strength  
and determination*



*Tough meetings?  
Abused estimates?*



Let's chat!

Trainings, Talks, Goodies

Daily Posts

Questions?

VictorRentea.ro

in @VictorRentea