SPRING, JAKARTA EE, CDI - GOOD PARTS

JAREK RATAJSKI

Software Developer, Wizard, Anarchitect

at Engenius GmbH

I WORK WITH JAVA EE SINCE ~2001

WITH SPRING SINCE 2006

- I remember `EJB-OSS`
- and huge xmls in Spring

At the moment I am making my hands dirty in about 15 various Spring and Java EE projects

I code for only few projects that are not Spring or Java EE based

I code for only few projects that are not `Spring`
or `Java EE` based

including one very critical application (netty based)

PERSPECTIVE

⚠ I like digging in production bugs:

PERSPECTIVE

⚠ I like digging in production bugs:

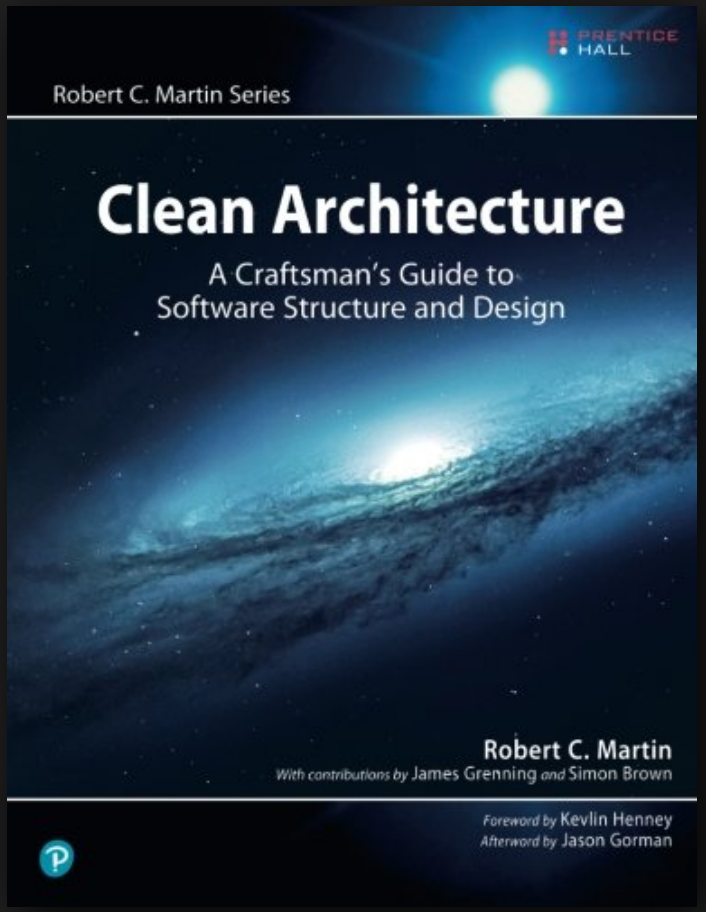
Concurrency, security, performance, heisenbugs, leaks

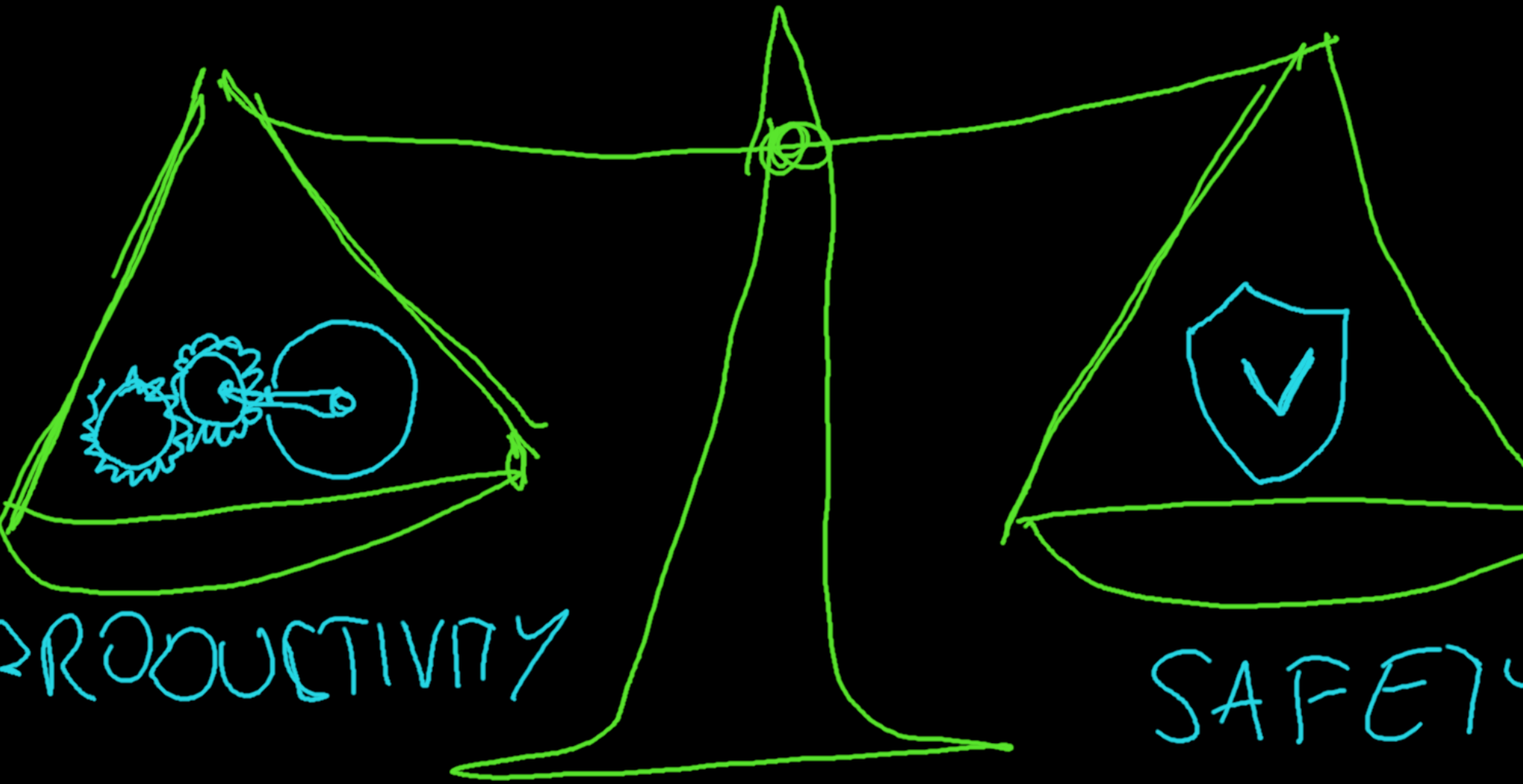
PERSPECTIVE

⚠ I like digging in production bugs:

Concurrency, security, performance, heisenbugs, leaks

Especially, not in my code


PRENTICE HALL
Robert C. Martin Series
Clean Architecture
A Craftsman's Guide to Software Structure and Design
Robert C. Martin
With contributions by James Grenning and Simon Brown
Foreword by Kevlin Henney
Afterword by Jason Gorman


PRODUCTIVITY
SAFETY

Safety:

- not many pitfalls bugs
- safe refactor (without bugs)
- easy to test
- meaningful (trustful) tests
- easy to introduce new team members

SAFETY

Type System

Tests

Typical `pitfall`

```
Connection conn = ...
    conn.init();
    conn.read(); //remember to call init before read
```

Typical `pitfall`

```
Connection conn = ...
    conn.init();
    conn.read(); //remember to call init before read
```

- It is easy to put warning into documentation

Typical `pitfall`

```
Connection conn = ...
    conn.init();
    conn.read(); //remember to call init before read
```

- It is easy to put warning into documentation
- It is easy to remember about it in a simple program

Typical `pitfall`

```
Connection conn = ...
    conn.init();
    conn.read(); //remember to call init before read
```

- It is easy to put warning into documentation
- It is easy to remember about it in a simple program
- It is easy to forget ... when you have 8 developers working 8 hours a day for few months

The problem: Java EE, Spring

introduce many `pitfalls` and they do harm to the code and architecture


ME TALKING
ABOUT
SPRING

This talk is about this harm, how to avoid that, what is still good in those platforms and what kind of alternatives we have

10 years ago my answer would be - you need to read books about Spring, Java EE, before you use them.

BEANS

WHAT IS A Bean?

Not the: JavaBeans

Not the: `JavaBeans`

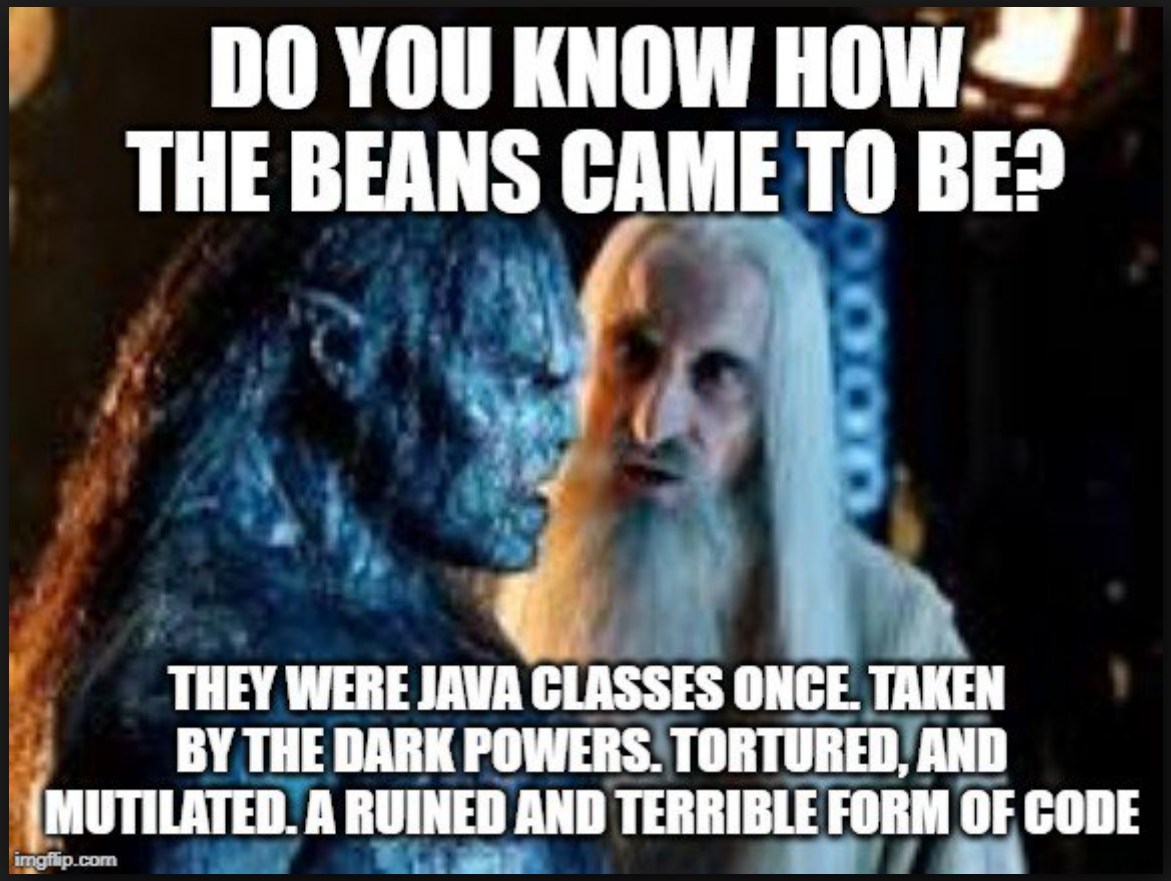
This is a closed topic, we don't go there anymore

Not the: `JavaBeans`

This is a closed topic, we don't go there anymore

I hope you do not write many getters and setters in 2020

SPRING BEANS, JAVA EE BEANS, CDI BEANS, JSF, JPA...



BEAN == MUTILATED, CRIPPLED CLASS(OBJECT) IN JAVA

BEAN == MUTILATED, CRIPPLED CLASS(OBJECT) IN JAVA

- Looks like normal object, but does not behave (like) normal

BEAN == MUTILATED, CRIPPLED CLASS(OBJECT) IN JAVA

- Looks like normal object, but does not behave (like) normal
-

BEAN == MUTILATED, CRIPPLED CLASS(OBJECT) IN JAVA

- Looks like normal object, but does not behave (like) normal
- Not instantiated by `new`

BEAN == MUTILATED, CRIPPLED CLASS(OBJECT) IN JAVA

- Looks like normal object, but does not behave (like) normal
- Not instantiated by `new`
-

BEAN == MUTILATED, CRIPPLED CLASS(OBJECT) IN JAVA

- Looks like normal object, but does not behave (like) normal
- Not instantiated by `new`
- there are special rules, limitations, conventions on use

BEAN == MUTILATED, CRIPPLED CLASS(OBJECT) IN JAVA

- Looks like normal object, but does not behave (like) normal
- Not instantiated by `new`
- there are special rules, limitations, conventions on use

  (pitfalls)

BEANS, BUT WHY DO WE HAVE THEM?

INJECTIONS


dependency injection - what is a DI?

In software engineering, dependency injection is a technique whereby one object supplies the dependencies of another object. A "dependency" is an object that can be used, for example a service. Instead of a client specifying which service it will use, something tells the client what service to use. The "injection" refers to the passing of a dependency (a service) into the object (a client) that would use it. The service is made part of the client's state.[1] Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

Wikipedia

Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

```
class MyService {
    private final DbRepo db;
    public MyService() {
        this.db = new DbRepo("jdbc://url")
    }
}
```

Do we have a dependency injection here?

```
class MyService {
      private final DbRepo db;
      public MyService(DbRepo  db) {
          this.db = db;
      }
}

MyService serviceProvider()  {
     var  db = new DbRepo("jdbc://url")
     return MyService(db)
}
```

Do we have a dependency injection here?

https://sites.google.com/site/unclebobconsultingllc/blogs-by-robert-martin/dependency-injection-inversion

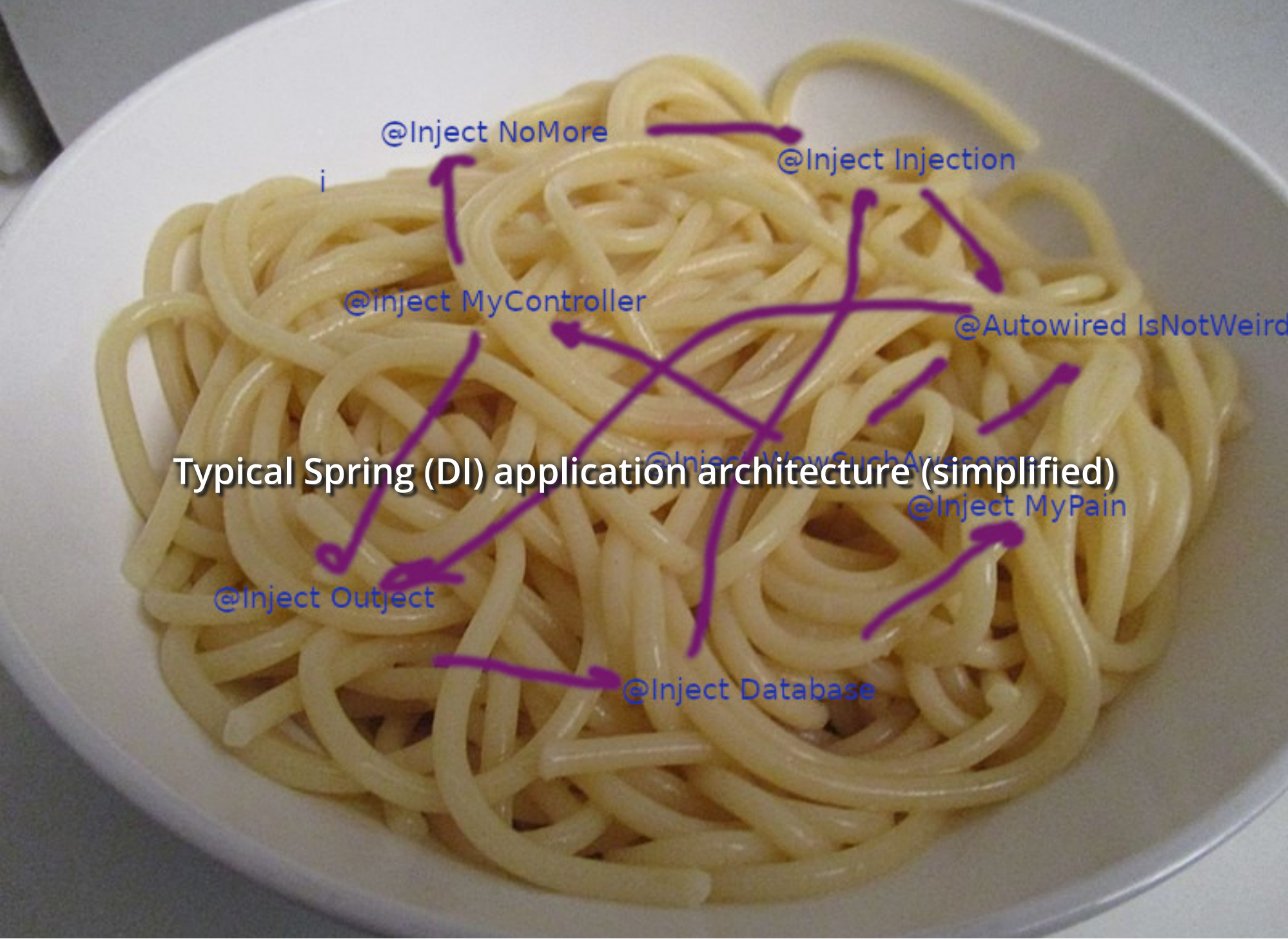
Dependency Injection doesn't require a framework; it just requires that you invert your dependencies and then construct and pass your arguments to deeper layers.

Framework (IoC container) lets you `inject` at a *small cost*

Framework (IoC container) lets you `inject` at a *small cost*

Framework (IoC container) lets you `inject` at a *small cost*

It is in fact technical debt - you will pay later



Typical Spring (DI) application architecture (simplified)

- Repository in Controller - no problem

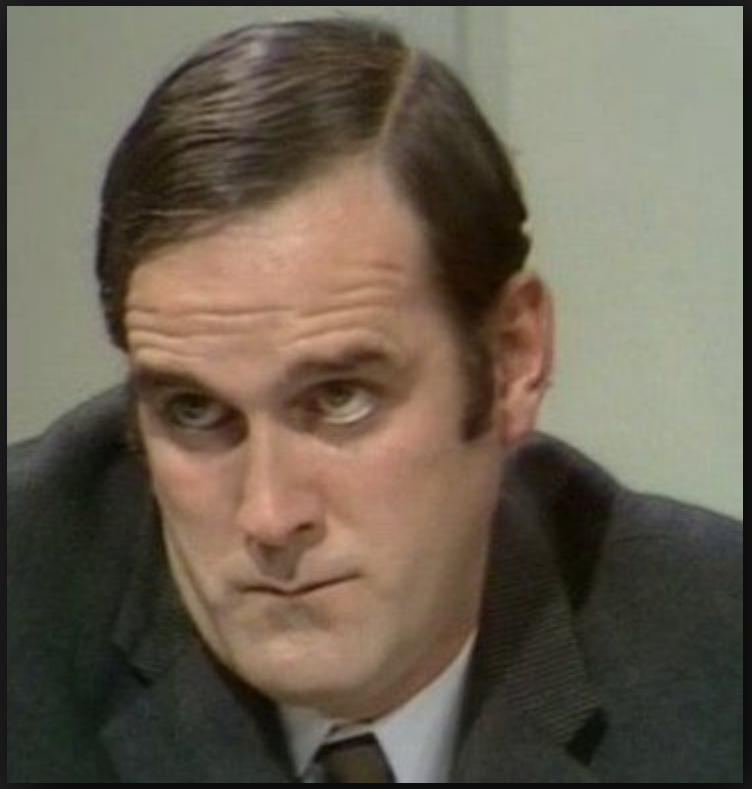
- Repository in Controller - no problem
- HttpRequest in Persistence layer - no problem

- Repository in Controller - no problem
- HttpRequest in Persistence layer - no problem
- Injecting anything anywhere - no problem

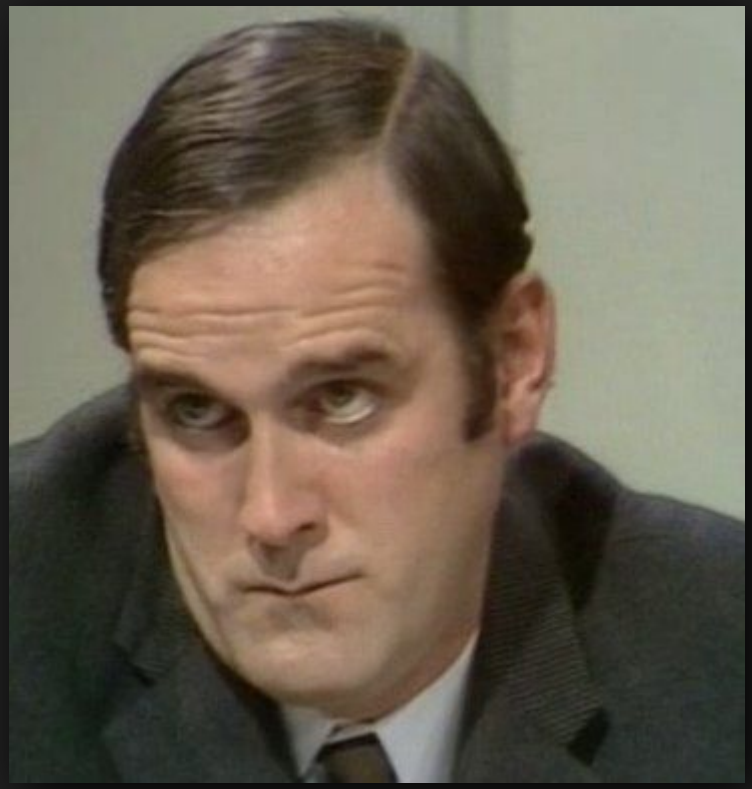
- Repository in Controller - no problem
- HttpRequest in Persistence layer - no problem
- Injecting anything anywhere - no problem
- Bean transferred diseases - gratis

Only bad developers do this

Only bad developers do this



Only bad developers do this



oh, really?

It is like GO TO

Sooner or later someone will make a shortcut an hour before DEMO

An it will stay like that - forever

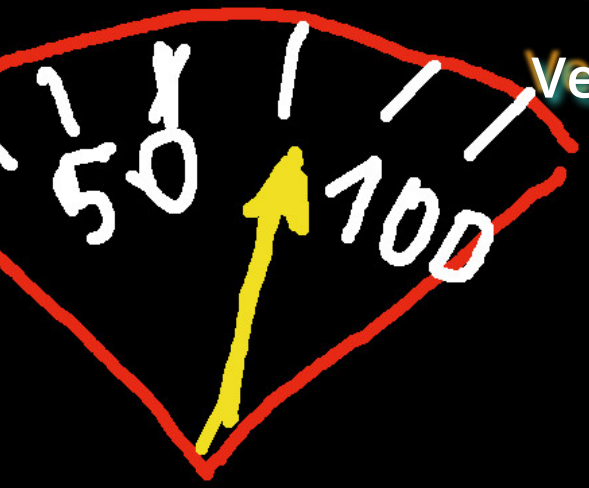
Because, the most important things in agile are:

Sooner or later someone will make a shortcut an hour before DEMO

An it will stay like that - forever

Because, the most important things in agile are:

Velocity and nice burndown charts



max(count(@Autowired in C
for all Classes C

LOBO

Level of Beans obscurity

(same for Jakarta EE - CDI `@Inject`, EJB `@EJB` ...)

It is even *pro containers* argument !

Would you write all those hundreds of injections manually?

Solution

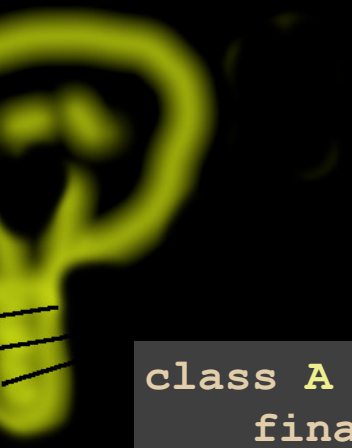
```
class A {
    @Autowired
    Xx;
    @Autowired
    Y y;
    @AUtowired
    Z z;
}
```

Step 0

```
class A {
    final X x;
    final Y y;
    final Z z;
    @Autowired
    A(X x, Y y, Z z) {
        this.x =x; this.y = y; this.z = z;
    }
}
```

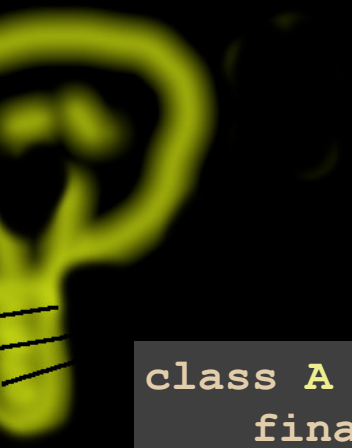
Step 1

(http://olivergierke.de/2013/11/why-field-injection-is-evil/)



```
class A {
    final X x;
    final Y y;
    final Z z;
    //@Autowired
    A(X x, Y y, Z z) {
        this.x =x; this.y = y; this.z = z;
    }
}
```

Step 2



```
class A {
    final X x;
    final Y y;
    final Z z;
    //@Autowired
    A(X x, Y y, Z z) {
        this.x =x; this.y = y; this.z = z;
    }
}
```

Step 2

Notice, this still works in spring

```
class A {
    final X x;
    final Y y;
    final Z z;
    A(X x, Y y, Z z) {
        this.x =x; this.y = y; this.z = z;
    }
}

class ServicesConfiguration {
    A getA() {
        return new A(this.getX(),this.getY(), this.getZ());
    }
}
```

Step 3 - finally without spring

BACK TO CODING SCHOOL:

BACK TO CODING SCHOOL:

Sections of repeating `new` can be extracted to methods

BACK TO CODING SCHOOL:

Sections of repeating `new` can be extracted to methods

Use factories / providers

BACK TO CODING SCHOOL:

Sections of repeating `new` can be extracted to methods

Use factories / providers

Too many arguements in constructor? Split a class in two (or three) (!)

PLENTY OF "BEANS" HAVE EXACTLY ONE IMPLEMENTATION

PLENTY OF "BEANS" HAVE EXACTLY ONE IMPLEMENTATION

Services, Controllers...

PLENTY OF "BEANS" HAVE EXACTLY ONE IMPLEMENTATION

Services, Controllers...

You don't need to make everything injectable/configurable

EXAMPLE (IN KOTLIN + VAVR)

```
data class StonesModule(
    private val seq: DbSequence = DbSequence(),
    val stoneRepo: Lazy<StoneRepo> = Lazy.of { StoneRepo(seq) },
    val stoneService: Lazy<StoneService> = Lazy.of {StoneService
    val stoneRest: Lazy<StoneRest> = Lazy.of {StoneRest(stoneSer
) //this is Kotlin uber constructor

// somewhere else
val myModule = StoneRepo( stoneRepo = Lazy.of{MyRepo()})
val service = myModule.stoneService.get()
```

"MANUAL" DI VS FRAMEWORK

Manual DI	vs	container IoC
small pain every day	<-->	no problem for months - then disaster
tree like structure	<-->	ball of mud (messy cake)
3 - 6 deps per class	<-->	5 - 18 deps per class (LOBO)


Manual DI

THERE ARE BEANS WORSE THAN SINGLETONS....

•
•
•

THERE ARE BEANS WORSE THAN SINGLETONS....

- `Request` scoped
-
-

THERE ARE BEANS WORSE THAN SINGLETONS....

- `Request` scoped
- `Session` scoped
-

THERE ARE BEANS WORSE THAN SINGLETONS....

- `Request` scoped
- `Session` scoped
- `ThreadLocal` based

THERE ARE BEANS WORSE THAN SINGLETONS....

- `Request` scoped
- `Session` scoped
- `ThreadLocal` based

Those are in fact *global variables*

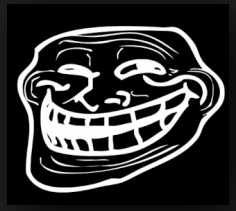
```
private C method1( A a, B b) {
 //uses a, b, and this. fields
  this.serviceX.method2(a); //does not use `b`

  this.serviceY.method3(b); //method does not use `a`
}
```

```java
@Component
@Scope(value = WebApplicationContext.SCOPE_REQUEST, proxyMode = ScopedProxyMode.TARGET_CLASS)
class TrollService {
    public A getA() {
        return a;
    }
    public void setA(A a) {
        this.a = a;
    }
    private A a;
}

@Service
public class ServiceX {
    @Autowired
    private TrollService trollService;

    void method2(A a) {
        trollService.setA(a);
    }
}
```



In spring based projects this is *normal*

(especially in Spring batch)

Broken *Local reasoning*

OUTCOME:

Many of small, reasonable changes can break your system

OUTCOME:

Many of small, reasonable changes can break your system

and tests are still green (because they test only mocks)

OUTCOME:

Many of small, reasonable changes can break your system

and tests are still green (because they test only mocks)

Sleep well, all your tests are green


CONTAINER INJECTIONS ?
NOT EVEN ONCE !
imgflip.com

… except …

- With *huge* codebases
  - Read: "Monoliths"
  - Read: "Enterprise"
- Enables a *tradeoff*
  - Developer discipline
  - Code coherence, simplicity, navigability

Corollary:

If you're seeing benefit from IoC, your codebase is *already out of control.*

WiXEngineering

Slaying Sacred Cows: Deconstructing Dependency Injection

Tomer Gabel

ASPECTS

@Transactional

When `@Transactional` does not work?

-
-
-
-
-
-

When `@Transactional` does not work?

- private method
-
-
-
-
-

When `@Transactional` does not work?

- private method
- public, but `this.call(...)`
-
-
-
-

When `@Transactional` does not work?

- private method
- public, but `this.call(...)`
- object instantiated with `new`
-
-
-

When `@Transactional` does not work?

- private method
- public, but `this.call(...)`
- object instantiated with `new`
- called in other thread (`parallelStream(), future`)
-
-

When `@Transactional` does not work?

- private method
- public, but `this.call(...)`
- object instantiated with `new`
- called in other thread (`parallelStream(), future`)
- troll aspekt (`@Trollsactional`)
-

When `@Transactional` does not work?

- private method
- public, but `this.call(...)`
- object instantiated with `new`
- called in other thread (`parallelStream(), future`)
- troll aspekt (`@Trollsactional`)
- missing jar on server

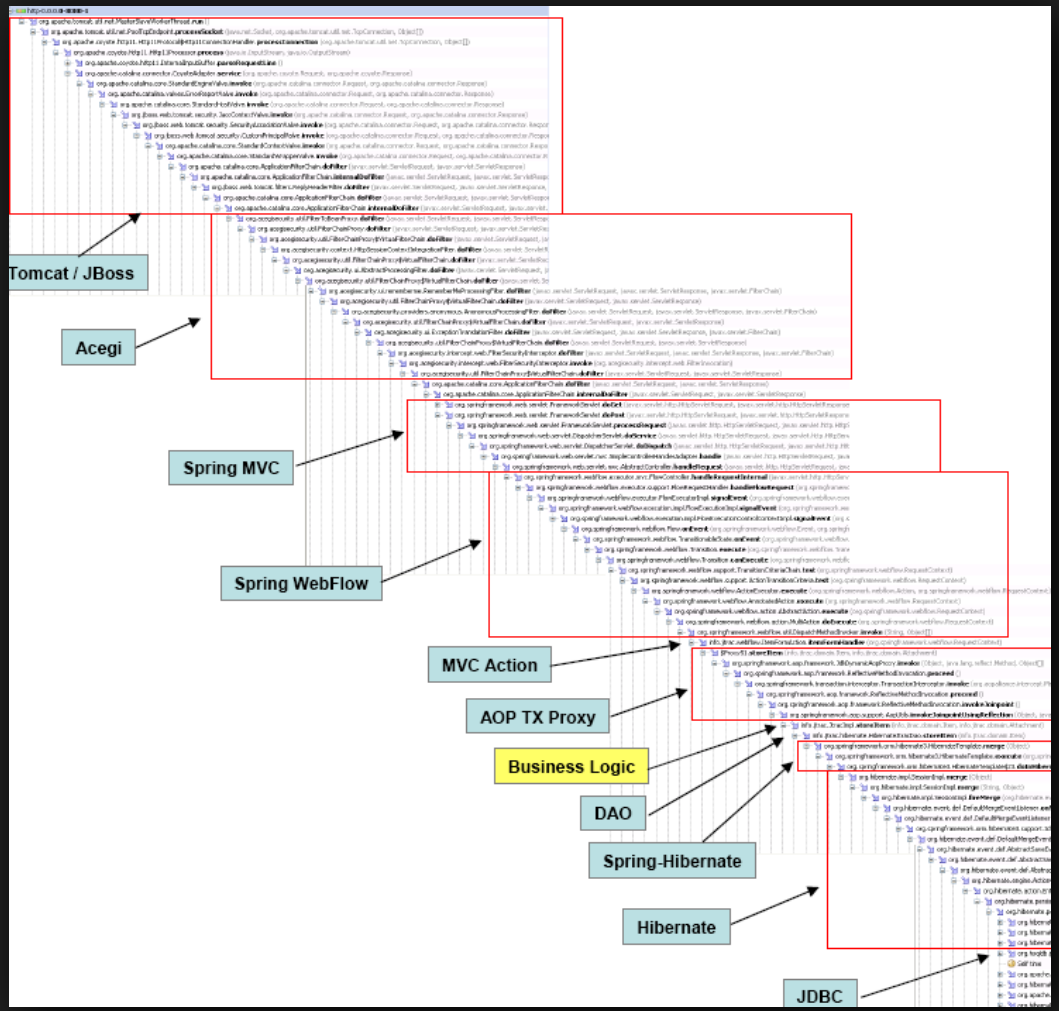
ADD JPA MAGIC ON TOP OF THAT

- yet another magic Beans
- managed
- detached
- dirty check
- proxy

ADD TRANSACTION ISOLATION LEVEL ASSOCIATED ISSUES

A.C.I.D.

Spring with `@Transactional` and JPA, and Database all together



If `@Transactional` does not work - where do you put a breakpoint?

All aspects induce similar problems:

- @Secured
- @RolesAllowed
- @Cacheable
- @Lock
- ...

All aspects induce similar problems:

- @Secured
- @RolesAllowed
- @Cacheable
- @Lock
- ...

Can your company accept that those aspects may be not active on production?

All aspects induce similar problems:

- @Secured
- @RolesAllowed
- @Cacheable
- @Lock
- ...

Can your company accept that those aspects may be not active on production?

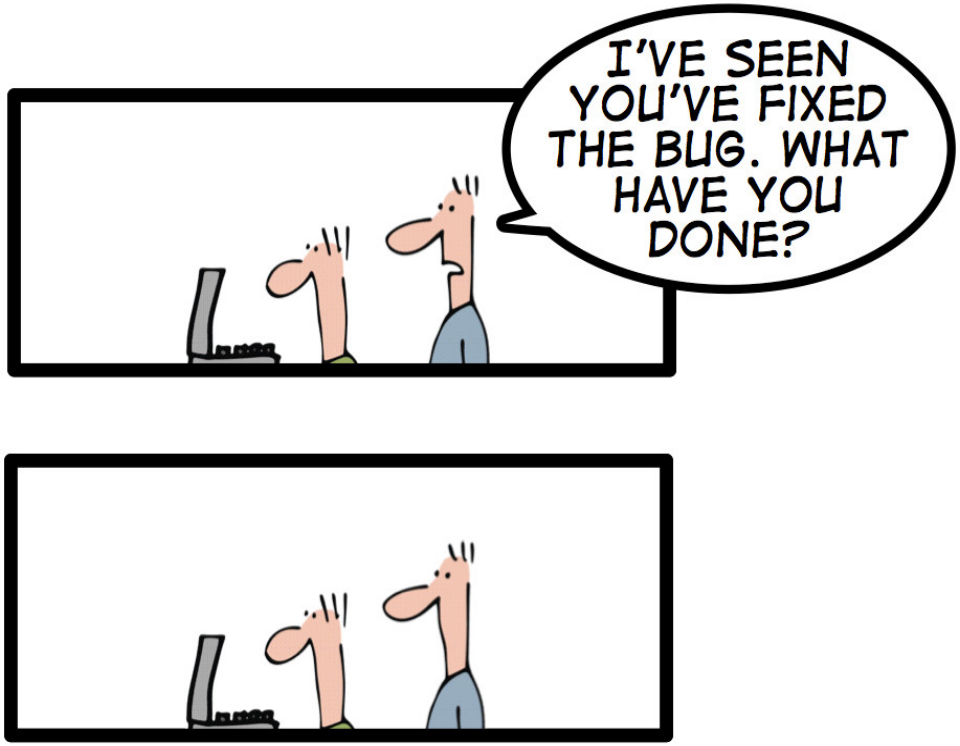
after small refactoring ?

All aspects induce similar problems:

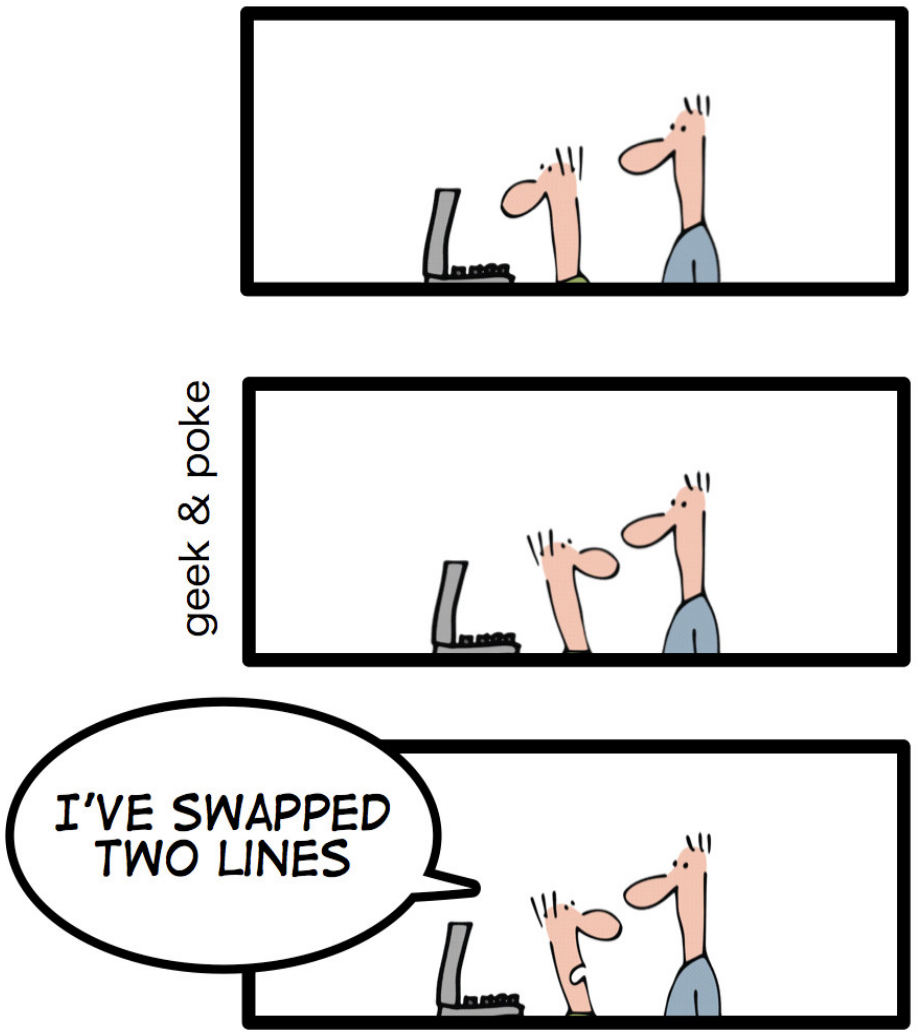
- @Secured
- @RolesAllowed
- @Cacheable
- @Lock
- ...

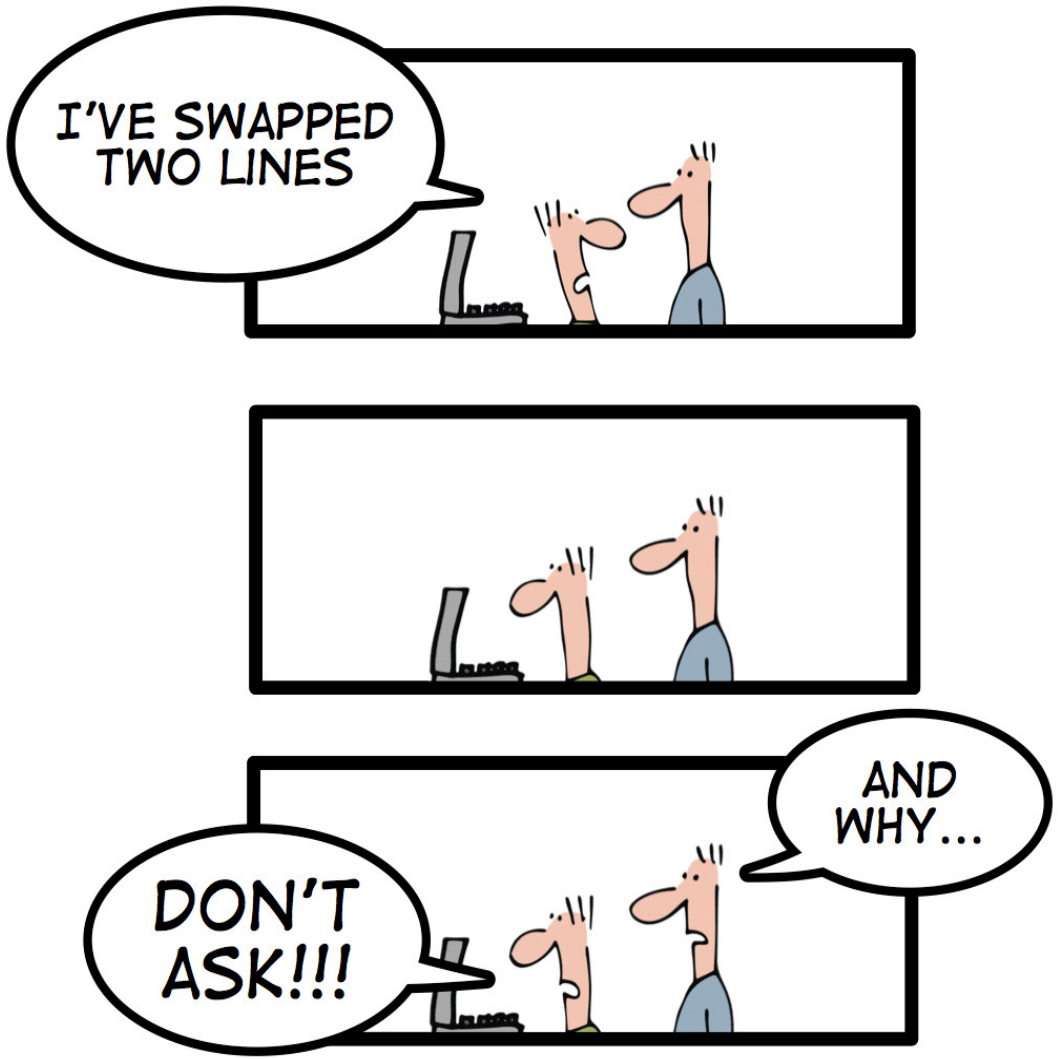
Can your company accept that those aspects may be not active on production?

after small refactoring ?



Those are just NOT edge cases


geek & poke
I'VE SWAPPED TWO LINES

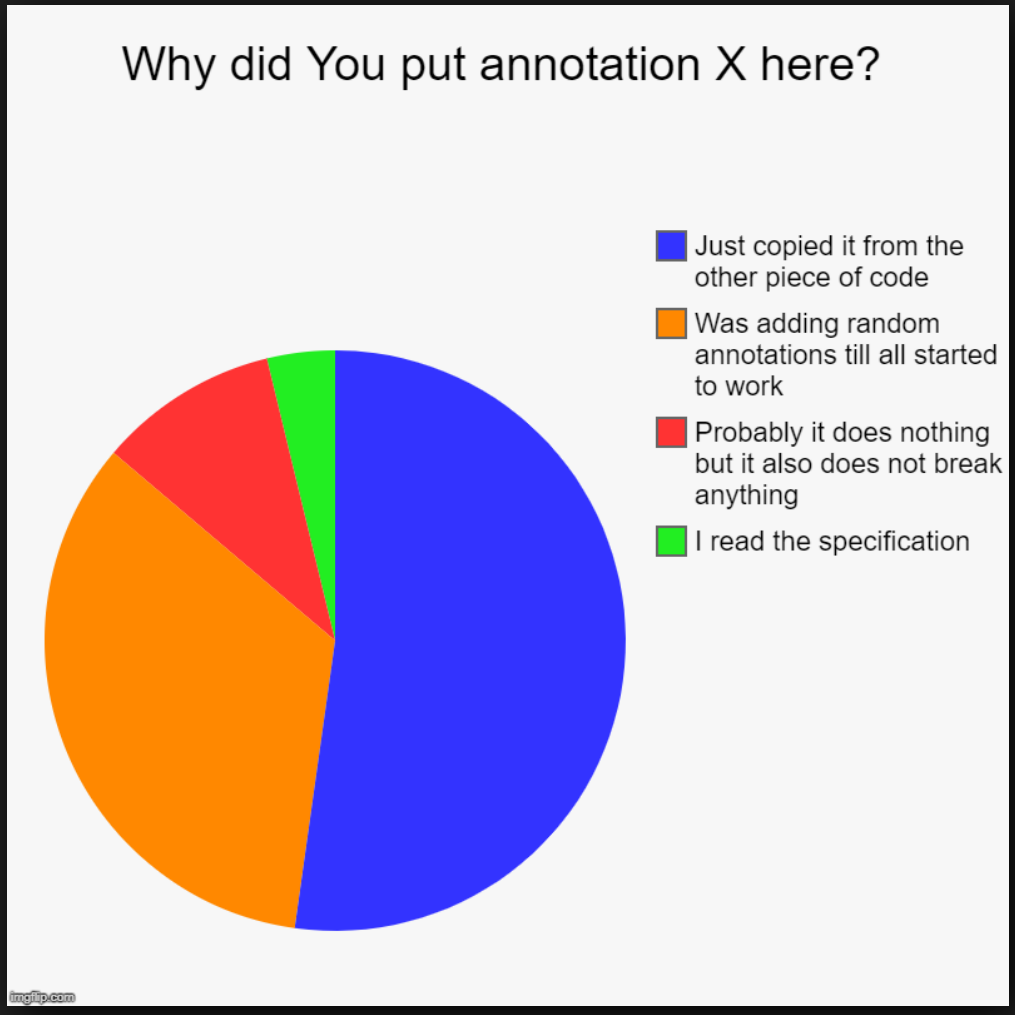


It happens more often than you think

Some of the problems (like async) are solved by another set of annotations

```
@AspectJ,
@PostConstruct,
@EnableAsync,
@EnableScheduling,
@NoRepositoryBean
```

Bean based development - a gentle introduction


Why did You put annotation X here?
Just copied it from the other piece of code
Was adding random annotations till all started to work
Probably it does nothing but it also does not break anything
I read the specification
imgflip.com

MAGIC IN CODE

MAGIC IN CODE

actually false (useless) definition

MAGIC IN CODE

actually false (useless) definition

THINGS, WE DO NOT UNDERSTAND

MAGIC IN CODE

MAGIC IN CODE

practical definition (v2.0 stable)

MAGIC IN CODE

practical definition (v2.0 stable)

THINGS, THAT DO NOT COMPOSE SAFELY

Magic is a feature with non-compositional semantics that succeeds in making the common case easy, at the cost of making the uncommon cases surprising, impossible, or ridiculously complex.

John de Goes (again)

MAGIC ON THE JVM

- Dynamic proxy
- Thread local
- Runtime reflection
- Instrumentation
- bytecode manipulation
- `Stringly typed` annotations

```
@Retryable
void myMethod () {

}
```

```
@Transactional
void myMethod () {

}
```

```
@Transactional
@Retryable
void myMethod () {

}
```

```
@Transactional
@Retryable
void myMethod () {

}
```

Is retry inside transaction or transaction inside retry?

```
@Transactional
@Retryable
void myMethod () {

}
```

Is retry inside transaction or transaction inside retry?

cache?, security? -> have fun

(hidden) cost of beans/aspect magic:

- Heisenbugs
- Paused development
- Unrealistic tests (aspects are not covered)
- Or Slow tests (with aspects)
- Overmocking (aka `Mocksturbation`, ...sorry)
- Fear of refactoring
- classpath / classloader disasters (on application servers)
- problem with new java versions (not in Spring)
- ugly architecture with *shortcuts*

HOW WE DEFINE NEW ASPECTS?

```
@Around("@annotation(Trollsaction)")
public Object doInTransaction(ProceedingJoinPoint joinPoint) throws Throwable {
    Tx tx = startDBTransaction();
    Object result = null;
    try {
         result = joinPoint.proceed();
        tx.commit();
    } catch(Exception e){
        tx.rollback();
    }finally {
    }
    return result;
}
```

simplified Transactional handler

WHAT IF ARE NOT USING ASPECT?

```
public  R doInTransaction(Supplier<R> inTransaction)  {
    Tx tx = startDBTransaction();
    R result = null;
    try {
        result = inTransaction();
        tx.commit();
    } catch(Exception e){
        tx.rollback();
    }finally {
    }
    return result;
}
```

```
public  R doInTransaction(Function<Transaction, R> inTransaction) {
    Tx tx = startDBTransaction();
    R result = null;
    try {
        result = inTransaction(tx);
        tx.commit();
    } catch(Exception e){
        tx.rollback();
    }finally {
    }
    return result;
}
```

Same pattern works for Security and other aspects

almost all aspects can be rewritten to function call with lambdas

You do not have to write your own:

```
create.transaction(configuration -> {
    AuthorRecord author =
    DSL.using(configuration)
       .insertInto(AUTHOR, AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
       .values("George", "Orwell")
       .returning()
       .fetchOne();

    DSL.using(configuration)
       .insertInto(BOOK, BOOK.AUTHOR_ID, BOOK.TITLE)
       .values(author.getId(), "1984")
       .values(author.getId(), "Animal Farm")
       .execute();

    // Implicit commit executed here
});
```

JOOQ Transaction handling

But not all aspects should be rewritten to functions:

- diagnostic,
- metrics

Are perfect example of aspects that are not very efficient if written in a functional way (noise)

WE DO NOT REALLY NEED BEANS

WE DO NOT REALLY NEED ASPECTS

IS SPRING USEFUL AT ALL?

SPRING IS DEAD LONG LIVE...

SPRING

```
class WisniaServer {
    fun start() {
        val route = router {
            GET("/hello", handle(::printHello))
            GET("/helloUser", secure(handle(::printlHelloForUser)))
        }

        val httpHandler = RouterFunctions.toHttpHandler(route)
        val adapter = ReactorHttpHandlerAdapter(httpHandler)
        val server = HttpServer
                .create()
                .host("localhost")
                .port(8080)
                .handle(adapter)
                .bindNow()
        println("press enter")
        readLine()
        server.disposeNow()
    }
}
```

Spring WebFlux (without beans)

SPRING WEBFLUX

- WebFramework
- Non-Blocking
- supports blocking
- Functional(*)
- Nice API
- No beans needed
- No spring context needed
- No aspect needed
- Simple testing

```
    void testProcessingSuccessful() {
          var result = WebTestClient.bindToRouterFunction(router)
                .configureClient().responseTimeout(Duration.of(defaultTimeout, ChronoUnit.SEC
                .build()
                .post().uri("/orders/process/$path")
                .exchange()
                .expectBody().returnResult().responseBody().toString()

        assertThat( result, is("ok"))
}
```

WebFlux can be mixed with classic spring (beans)

although this is magic

Standard WebFlux with Reactor is a total (hard) new thing to learn

Actually more Spring modules can be used in a clean way - without beans and spring context

WHAT ABOUT JAKARTA EE?

I valued it for being well documented consistent server framework

Great for year 2000

Then it started to be more Springy than Spring

I think only in Java EE projects you will see `@PostConstruct`

Most of applications servers can work as embedded now!

It is 2020, please DO NOT use standalone application servers

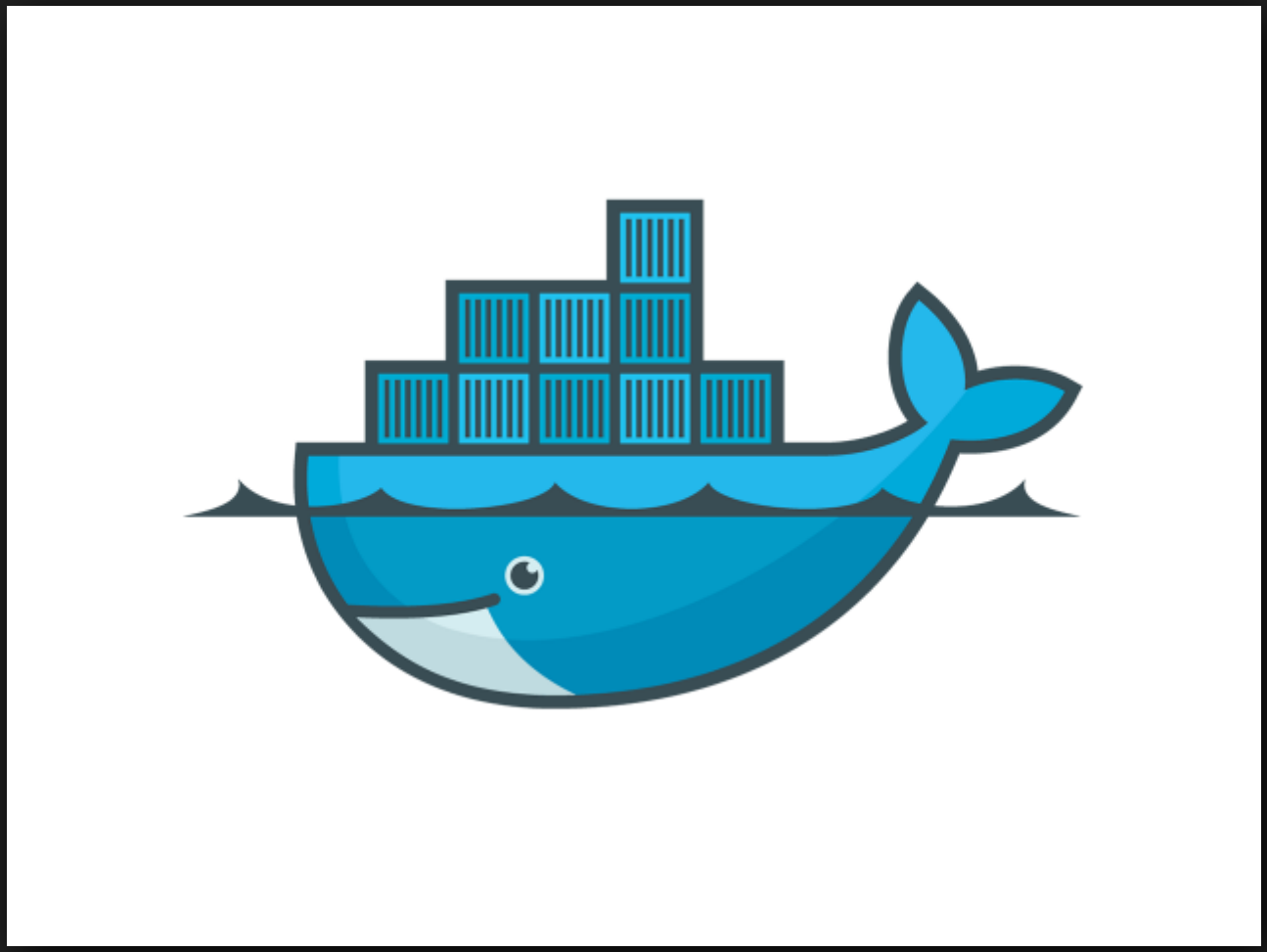
- problems with classpath, classloaders
- problems with jvm versions
- noise in logs
- tough to test (arquillian)
- configuration issues
- jvm params hell

Make jar not war

Make jar not ear

Need containers?

We have plenty of them already



BLOWN EGG PATTERN

```
my-app-core:
     services, business logic, tests - clean java
my-app-ee:
    java-ee wrapper: jax-rs, datasources etc.
```

```
@Path("/order")
public class Orders {
  @PersistenceContext
  final EntityManager em;
  @GET
  @Produces(MediaType.APPLICATION_JSON)
  public List<String> showOrders() {
    return new OrdersService(em).listOrders();
  }
}
```

Same pattern applies for Spring

JAKARTA EE IS FAR AWAY FROM INITIAL IDEA

Initially:

- better CORBA,
- remoting,
- distributed transactions support,
- resource (RAM) friendly (more apps on a single JVM)

Today: Single application on a single application server that is not using any distributed transactions

SOLUTIONS

Bill G flicker image (https://www.flickr.com/photos/billerr/1814657036)

Bill G flicker image (https://www.flickr.com/photos/billerr/1814657036)

Baby steps

STEP 1

- Drop application servers
- Make Jar not War

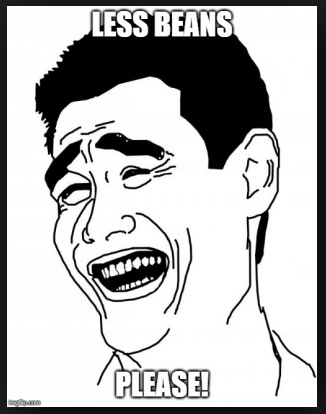
(Spring Boot is good for a start)

STEP 2

- Hold your Beans
- Use Beans where really needed
- (example -> only Jax-RS annotations and `@Controllers`)
- Use only constructors for DI

STEP 2

- Hold your Beans
- Use Beans where really needed
- (example -> only Jax-RS annotations and `@Controllers`)
- Use only constructors for DI



STEP 3

(fun starts)

Drop JPA

Try: JOOQ, QueryDSL, JDBI and alternatives

STEP 4

Learn altrernative web/rest frameworks

- akka-http
- SparkJava
- javalin
- ktor
- ....

STEP 5

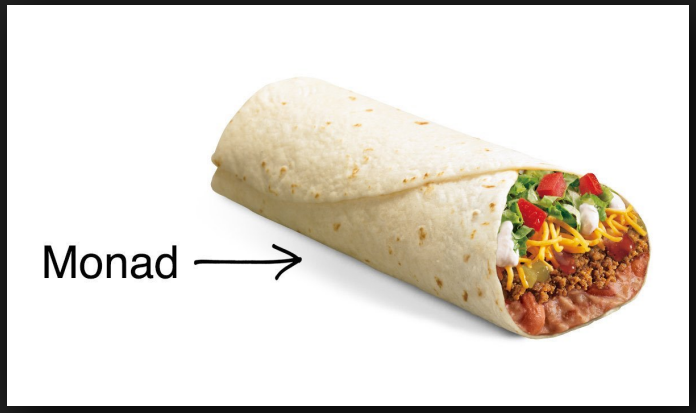
Try functional programming

Transaction is a monad

STEP 5

Try functional programming

Transaction is a monad



```kotlin
class Transaction<A> (private val action : (Handle) -> A ) {

    fun run(dbi : Jdbi) : A = dbi.withHandle<A, RuntimeException>(action)

    fun <B> map ( f: (A)->B)  = Transaction {hande ->
        f(action(hande))
    }

    fun <B> flatMap( f: (A)->Transaction<B>)  = Transaction {handle ->
        f(action(handle)).action(handle)
    }

    companion object {
        fun <T> pure (obj:T) = Transaction {
            obj
        }
    }
}
```

Experiment (future)

Instead of writing:

```
class Hasiok {
    @Resource
    val jdbcConnection: Connection

    @Transactional
    @Secure
    @Cacheable
    @Retryable
    fun f(p:P) {
        //code
    }
}
```

You may write this:

```
class  Hasiok {
    private val  f =  {jdbcConnection:Connection ->
            {p: P ->
            //code
        }
    }
    val enterprisyF = Nee.pure(
        secure
        .and(retryable)
        .and(cacheable)
        .and(transactional), f)
}
```

OPINION

- Spring is battle tested has great docs - no other platforms in a JVM world is close to that (Akka, Lagom, ZIO etc) ✓
- Every java developers knows Spring or Java EE on a shallow level ✓
- Java developers are unaware of bean associated complexity ✓
- We can still benefit from / use Spring and Java EE - while minimising use of Beans ✓

Thank you

@jarek000000

Sources:

- Adam Warski 2017 - The Case against annotations
- Tomer Gabel - Slaying sacred cows
- https://sites.google.com/site/unclebobconsultingllc/blogs-by-robert-martin/dependency-injection-inversion
- From Spring Boot Apps to Functional Kotlin Nicolas Frankel https://www.youtube.com/watch?v=f6a78mCrSeE