Contents

[Core Java 2](#_Toc512183383)

[Revisiting Effective Java in 2018 (Edson Yanaga) 2](#_Toc512183384)

[JDK 9, 10, 11 … / What’s next 4](#_Toc512183385)

[Architecture / Design / Clean code 6](#_Toc512183386)

[Du DDD dans mon Legacy – Thomas Pierrain, Bruno Boucard, Jérémie Grodziski 6](#_Toc512183387)

[Mutation Testing (how to evaluate the quality of your unit tests)s 7](#_Toc512183388)

[Pourquoi avez-vous besoin de la Clean Architecture ? 7](#_Toc512183389)

[New frameworks/hands on feedback 7](#_Toc512183390)

[Feedback from JHipster on upgrading to Spring Boot2 7](#_Toc512183391)

[Spark – Optimisation de performance 8](#_Toc512183392)

[Lazy Java 10](#_Toc512183393)

[Swagger 2 est mort, vive OpenAPI3 10](#_Toc512183394)

[Hardcore / Low level Java 11](#_Toc512183395)

[ByteCode Pattern Matching 11](#_Toc512183396)

[Java.lang.invoke / whisper to the JIT for speed gain 11](#_Toc512183397)

[Organisational/Process 12](#_Toc512183398)

[Asynchronous decision making, why and how ? 12](#_Toc512183399)

[Others 12](#_Toc512183400)

[Java Vulnerabilities 12](#_Toc512183401)

# Core Java

## Revisiting Effective Java in 2018 (Edson Yanaga)

**Quickly**: 3 hours live coding by the author of the well known ‘Effective Java’ book, demonstrating a panel of best practices in Java (>=8). While there’s no innovation here, it’s really great to have some guideline from a really expert guy.

**What I will keep in mind:**

* Focus on design immutable objects whenever possible
* Keep in mind than immutable objects offer more useful possibilities than unmodifiable objects. (guava framework offers some helpers to build immutable collections)
* Prefer using strategy to template pattern (with the help of functional interfaces/lambdas)

**Main topics of the live coding session:**

**Factory methods**

* Prefer factory methods (pubic static PhoneNumber of(…) ) to public ctors
  + It offers the possibility to reuse instances (basic use case : you build two objects with the same arguments, you return the same instance), while it is not possible to do so when exposing a public ctor
  + Prefer using the builder pattern when more than 3 attributes in the ctor / factory method
* Check the arguments before building the object
  + Use jdk8 checkers (ex: Objects.requireNonNull)
  + Guava framework offers a richer set of precondition checkers and handles primitive types (thus avoiding unnecessary boxing when using jdk8 checkers on primitive types)
    - Cf Guava framework for precondition checkers (see for ex: com.google.common.base.Preconditions.checkArgument)
* On the implementation of Object.equals : use Objects.equals to compare objects (jdk8 method)
* On the implementation of Object.hashcode :
  + Use Objects.hash (from JDK)
  + For immutable objects, initialize hash in ctor to be more performant
* On the implementation of Object.toString:
  + Guava provides some helpers to implement toString method (see MoreObjects.toStringHelper from guava framework)
* When implementing Comparable: as a reminder, Java 8 Comparatos do quite well the job (no use to use Guava’s framework anywork since Java8)

**Builders**

* Use constructor with mandatory arguments
* Optionals arguments as methods
* Use checkState (from guava) to check the arguments before building instance (to prevent building an object with inconsistent arguments)

**Singleton**

* Use Enums : best way to implement Singletons, if needed (not recommended to implement Singletons nowadays …)
* Only disadvantage : no inheritance (minor disadvantage)

**Strategy pattern**

* Old way to implement (before java8) is done with enums with abstract methods
* New way with java 8 => replace with java 8 functions (java.util.function) and use lambdas
* To enhance readability, encapsulate lambas in a package protected class & use method reference in Enum

**Try with resources**

* Nothing to say here … Use try with resources whenever possible …

**Functional interfaces**

* Favor strategy over template pattern (more readable/cleaner code)
* Strategy pattern using functional interfaces / lambdas

**How to handle exceptions inside lambdas ?**

* Check vavr.io framework

**‘Lombok-like’ frameworks (for code-generated hashCode/equals/toString/builders …):**

* Google auto
* Immutables.github.io
* Lombok

**Lambda & Functional Interfaces**

* Use method references when possible to increase readability

**Recap of Optional**

* Not serializable
* Not compatabile wih java beans spec
  + Make field as not optioal
  + Getter returns Optional.ofNullable

**Usage of var (java 10)**

* When to use ? to improve readability when instantiating objects
  + Example: var map = new VeryLongNamedClass….
* When not to use ? when affecting result of a method (you’re losing the return type defined in the signature) => less readable
  + Example : var result = someMethod(); // you have to go into the method signature to get the return type

**How to make immutable objects (make objects immutable whenever possible!!)**

* Defensive copy
* Guava provides helpers for defensive copies (ex: ImmutableList.copyOf)
* One (among many) advantage of immutable objects : they are thread safe by design

## JDK 9, 10, 11 … / What’s next

**Speeker : Simon Ritter (deputy CTO from Azul !)**

**Main topics:**

* Backward compatibility no more guaranteed
  + Until Java8, backward compatibility was ensured (no removal of APIs): you can build a Java6 project with a Java8 JDK…
  + From Java9, deprecated APIs will be removed
    - Examples:
      * Corba
      * SOAP XML bindings

* New JDK release strategy
  + Lifecycle/Versions
    - New version of JDK every six months
    - Features will be included only when ready
    - JDK Version number will now be consistent
      * Naming convention : FEATURE.INTERIM.UPDATE.PATCH
  + Oracle is switching to LTS model
    - LTS release every 3 years
    - JDK11 will be the next LTS
    - **Impact**: **only Oracle customers will be able to use LTS versions !!**
  + Less versions supported : no more 32 bit binaries/ ARM binaries . Only Windows,   
    Linux, MAC, Solaris SPARC only (all 64 bits)
* More open-source / reduce discrepancies between OpenJDK and Oracle JDK
  + **Licensing**
    - Oracle Binary Code License (traditional)
    - New OpenJDK Binary (GPL2 with CPE license)
  + At target, no functional differences between OpenJDK and Oracle JDK
    - Only bug fixes or security fixes will be included into Oracle JDK, not merged into OpenJDK minor versions
  + Closed source parts of the JDK will be open sourced:
    - Flight Recorder
    - Mission Control
    - Others (Webstart…)
* Content of JDK10 (JSR 383)
  + Local variable type inference (var)
  + Parallel full GC for G1
  + Application Class-Data Sharing
  + Experimental Java based JIT Compiler (Graal)
  + Root certificates default set (cf OpenJDK merge)
  + Garbage Collector interface enhancements
  + Thread local handshakes
    - Execute callbacks on threads without performing a global VM safepoint
  + Other
    - Optional.orElseThrow
    - JVM now more Docker-Container aware
    - com.sun.security.auth removed & more packages removed
* Content of JDK11
  + Only for Oracle customers (no free version from Oracle)
  + JEP 309
    - Dynamic class file constants (interesting mainly for language developers
  + JEP 318
    - New GC : Epsilon (a garbage collector which doesn’t collect garbage…)
      * For GC developers purposes (benchmarking purposes …)
    - CORBA modules removed
  + Local variable syntax for lamba parameters !
  + 36 new APIs

**For information : Other OpenJDK Projects**

* **Amber**
  + Local variable type inference (merged in JDK 10)
  + Local variable syntax for lambda
  + Enhanced enums
    - Generic enums with type parameters
    - Lambda leftovers
      * Single underscore for unused parameters (\_ becomes a reserved keyword)
  + Raw string literals (using single backquote)
  + Pattern matching
    - Type matching in switches (using different types in switch cases):

Ex:

Case Integer i:

* + - Use lambas in switches

Ex:

Int numLetters = switch (day) {case MONDAY, FRIDAY, SUNDAY -> 6

* **Valhalla**
  + Dealing with primitive types in generics
  + Framework for Value types
* **Loom**
  + Used to make concurrent programming simplier
  + By using fibres
* JVM level threads (green hreads), no OS level threads
* Using ForkJoinPool scheduler
* Benefits : more light-weight than OS-threads
* **Metropolis**
  + Rewrite JVM in Java (using Graal compiler as significant input)
* Merchandising from the speeker
  + Check Zuul JVM on <http://www.azul.com:downloads/zulu>

# Architecture / Design / Clean code

## Du DDD dans mon Legacy – Thomas Pierrain, Bruno Boucard, Jérémie Grodziski

**Quickly :** interactive refactoring session demonstrating clean code (example taken from Emilie Basch famous coding katas).

It was nice to see Bruno Boucard in action ;)

**Book to read (must read!!):**

Michael Feathers – Working Effectively with Legacy Code

DDD by Eric Evans

Refactoring legacy code (Martin Fowler)

**Best pratices**

* Renommer les variables / paramètres pour s’aligner avec le domaine fonctionnel
* Utiliser des mocks/stubs pour couper les dépendances avec des librairies ou systèmes externes (db, ws …)
* Remonter la déclaration des variables locales au plus proche de leur utilisation
* Single Responsability

**Some DDD Patterns**

* Pattern Entity
  + Has identity and lifecycle
  + Mutable
  + Identifier equality
* Value Type (Value Object)
  + No itdentity
  + Immulable
  + Equality by properties
* Aggregates
  + Collection of entities and value types treated as a whole
  + Aggregates needs invariants (with specific business rules)
  + If no invarariants => it is a collection
* Closure of operation
  + An operation whose reurn type is the same as the tpe of its implementer
    - Eg Coach otherCoach = coach.addSet(seat) 🡺cf defensive copy
* Hexagonal architecture
  + => cf Alistair Cockburn
  + To protect domain code
  + Components:
    - Center : domain
    - Infra code (Spring boot, etc…)
    - Interface (port in domain namespace) & Adapters (in infra code)
  + Configurable dependencies
  + Three steps for building an application using hexagonal architecture
    - Instantiate ‘go out’ adapters (ports)
    - Instantiate hexagons (domain code)
    - Instantiate ‘go in’ adater(s) (to access hexagon)
* Do not pollute domain with framework specific code (like ORM annotations, such a JPA/ Hibernate/Spring)
* Persistence should be handle in adapters, not in domain (topic to be discussed …)
* Domain code is the real value => adapter / infra code is more likely to change / domain code not changing
* Use ubiquitous language in domain code

## Mutation Testing (how to evaluate the quality of your unit tests)s

**Quickly**

A demo from Sarah Buisson (Xebia), demonstration PiTest (framework for mutation testing).

**What is mutation testing?**

Mutation testing is a technique to modify dynamically the code, and applying the unit tests.

If the unit tests are still green after applying a mutation in the code, that means that the unit tests are not sufficient.

**More infos :**

See pitest website

See Sarah Buisson github (github.com/sarahBuisson)

## Pourquoi avez-vous besoin de la Clean Architecture ?

**Speakers :**

Xavier Balloy @xavierballoy

Julien Hamon @hamonjulien1

**Quickly :**

Another speech on the benefits on separation of concerns. Feedback from two guys that implemented a mobile app at Axa.

They applied ‘Clean Architecture’ and were happy to see the benefits of it.

**Nota bene:** the concepts are quite the same as hexagonal architecture.

**More infos**

Check the reactive pattern

# New frameworks/hands on feedback

## Feedback from JHipster on upgrading to Spring Boot2

**Quickly**

Feedback from JHipster CTO, on the difficulties and benefits on migrating from Spring Boot 1.5 to Spring Boot 2.x

**Content**

* Spring Boot2 supports Java10 (Spring Boot 1.5 stops to Java8)
* Dependencies
  + Ehcache uses new Java8 Dateie API
  + Elasticsearch does not work any more in embedded mode
  + Spring Boot2 uses no CGLIB by default (no more proxy JDK by default)
    - No more useful to use interfaces for sping beans
* Lot of changes in yaml properties …
* Spring-boot-properties-migrator does the job to migrate automatically the config (**must use to migrate to Spring 2)**
* Migration de Spring Boot Actuator  
  **Warning**: the output format is different … Lots of impact for the monitoring tools that uses actuator
* Spring Data
  + Usage of Java 8 Optional has been generalized (impact with dependencies which do not support Optional)
  + Changes in API signature
* Spring Cloud
  + No release version yet

**New in Spring Boot 2 !!**

* Spring Webflux
  + To do “reactive” code
  + Using JPA => not very interesting because not very performant
  + Some use cases are interesting (example : stream data from a blob in a DB)
  + Support for MongoDB/CouchBase/Cassandra for data stream
* Spring Security
  + Support for reactive applications (security is no more handled in a local thread)
  + OAuth2 integrated in Spring Security 5
  + OIDC support for login
  + AuthenticationManager must be explicitely created
* Micrometer
  + New lib for monitoring
  + Activated by default (**Warning :** beware for performance overhead)
* Spring Boot banner can now be realized with animated GIF (not very useful ; ) )

## Spark – Optimisation de performance

**Quickly**

A quick talk by two guys from Xebia on some strategies to optimise Spark applications.

**Remark**

All the example were based on Spark 2.2.0 (thus some topics like Datasets do not exist in Spark 1.6.x)

**Content**

* Understanding the concepts of narrow dependencies vs wide dependencies and the impact on the shuffling of the data
  + map, filters, flatMap are usually transformations with narrow dependencies (lower shuffling)
  + groupByKey, distinct, join are usually transformations with wide dependencies (higher shuffling)
  + **It is usually more efficient to perform a reduceByKey rather than a groupByKey to reduce the shuffle**
* **Use KryoSerializer** to optimize the serialization cost.  
  Drawback: you must explicitely register the classes to serialize in KryoSerializer.
* When using disk persistence, don’t forget to compress the RDD (spark.rdd.compress)
* Keep in mind it is possible to use off-heap persistence
* Note : starting from Spark 2.0.0, Dataframe is a type alias of Dataset<Row>
* Limit the usage of Map structures inside a RDD (waste of memory)
* **On storage layer**
  + You have the option **inferschema** to deduce the informations from a source
  + Use parquet or another appropriate format for storing data (do not store in raw data)
  + You can sort data within partitions (using **sortWithinPartitions**)
* On cluster management layer
  + **Dynamic Allocation :** Starting from Spark 2.0.0, you can use dynamicAllocation to allocate executors on the fly
  + **Speculation** : spark can detect and relaunch slow running tasks
  + **Level of Parallelis :** Don’t forget to tune the level of parallelism on an executor
* **Other topics (**did not had time to get an idea on all these topics)
  + Multiplexing (splitting rdd and increase the level of parallelism)
  + Broadcast join (cf spark.sql.autoBroadcastJoinThreshold)
  + Partial broadcast
  + Bucketing
    - bucketBy / saveAsTable (only in spark 2.0 ?)
    - !! uses metastore (!= hive bucketing)
* To have in mind : no UDF in Python 🡺peformance overhead due to JVM to Python interpretor on executors

**Topics to investigate on:**

* Spark SQL AST
* Tungsten engine

## Lazy Java

**Quickly**

Speech from Mario Fusco (italian java champion), have some fun with lazy evaluation/functional programming, and how this paradigm can help a lot in some use cases.

**Content:**

* Benefits of java functional interfaces for lazy evaluation (in particuliar, Supplier)
  + For example, in log4j, use supplier to evaluate the argument to log only when needed (note from me : in theory, it works, it practice, Remi Forax showed that it’s not true)
* Difference between standard recursion and tail recursion (TCO - taill call optimiaztion)
  + Less use of stack
  + Example of usage of TCO : Scala
* See trompoline pattern (note from me: check gihub.com/functionaljava for example of trampoline in Java)
* Reader monad (for implementing lazy dependencies injection)
  + To implement lazy injection (by using function application & composition)

## Swagger 2 est mort, vive OpenAPI3

**Quickly :**

Speech describing the benefits on switching to OpenAPI3 (> Swagger2)

**Content**

* **OpenAPI 3**
  + Cf **openapi-map.apihandyman.io**
  + <https://editor.swagger.io>
  + GFM => Common Mark
  + Gestion des versions en SEMVER
  + JSON Schema (Draft 4 => Draft 5 -Wright Draft 00)
  + OIDC Support
  + Server configuation
    - Ajout de requestBody dans le yaml
    - Composition / Polymorphisme avec oneOf
  + Webhook support
  + links
* **moustache => ?**
* **generator de code :**
  + **swagger-codegen**

cf <https://github.com/OAI/OpenAPI-Specification/issues/1466> pour prochaines fonctionnalités

Limitations :

* Traits & resource types
* Templating des components

# Hardcore / Low level Java

## ByteCode Pattern Matching

**Evgeny Mandrikov (SonarSource, @\_godin\_)**

JaCoCo submitter

Not Synthetic methods (like default tor…)

Synthetic methods:

Ex : when nested classes

Lambda

Assert

Switch on String 🡺 using hashode(), not String

Exception tables

Try with ressources

Javac8 improvement on null checking to improve bytecode

Java11 compiler enhacement on null cheking

Eclemma

Groups.google.com/forum/jacoo

## Java.lang.invoke / whisper to the JIT for speed gain

Rémi Forax – Java Champ

Amber & valhalla

Coût d’un logger disabled

<jmh

VM oent trust final fieldsbecause of serialization

Java.lang.invoke

Better than reflection

Do he security check once

No default boing

Used in lambda

Used in sring concatenation in 9

invokeExact =< pas de boxing

partial application

GuardWithTest

Empty method handle in jdk9

ciField.cpp (trust\_final\_on\_static\_fiields)

lambda => classe anonyme pour la JVM

log4J => pas perfomant

probleme cham volatle

replae volaile with SwitchPoint

SwitchPOint triggered🡺code JIT invalidé

Github.com/forax/beautiful\_logger

Github.com/forax/exotic

# Organisational/Process

## Asynchronous decision making, why and how ?

Bertrand Delacrétaz – Adobe (&board Member, Apache Software Foundation)

* Cf Paul Graham (<https://s.apache.org/ms>)
  + Planning arisan vs artisan manage
* Cordova-discuss (cf github.com/apache/cordova-discuss)

Canal partage asynchone

Méthode de créaton de consensus

Outil de suivi de cas pour les décisions

<https://pinboard.in/u:bdelacretaz/t:collaboration>

<https://github.com/bdelacretaz/opendev-channel-requirements/>

# Others

## Java Vulnerabilities

@spoole167

parseDouble eploit

smissing URL.toURI() (find CVE ?)

TrustManager overriding

<https://cve.mitre.org>

CVSS 3.0 calculator

Oracle

PSIRT Blog

IBM

OpenJPEG vulnerability

CVE 017 5638 (Equifax) Apache struts vulnerability

Orace serialization security guidelines

EoinWoods1/secure by esign security design principals

* Checkmarx