L'immutabilité

Ce super-pouvoir contre les bugs !

Inspiré d'une histoire vraie

Sébastien Laoût

mardi 26 avril 2022

Ubik
INGÉNIERIE

Qui suis-je ?

Sébastien Laoût

Chez Ubik Ingénierie depuis 10 ans
Développeur full-stack depuis 14 ans
Java / TypeScript
Principalement sur de l'e-commerce
Actuellement Tech Leach chez Adeo Services
Chargé de la maintenabilité du projet

Décollage d'Apollo : 4 pièces mobiles

[...]which left only four moving parts to go wrong in the entire ascent engine[...]
APOLLO 11 Owner's Workshop Manual - Haynes

Plan

Problèmes

Solution

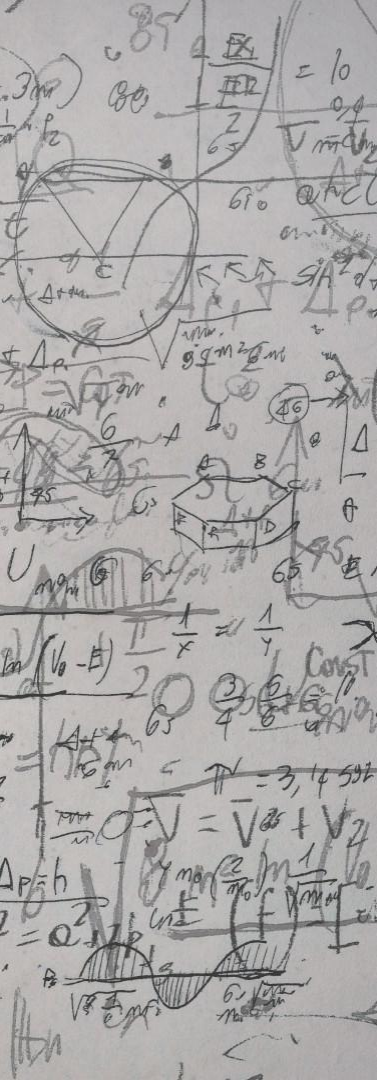
Implémentation

Refactoring

Et après ?

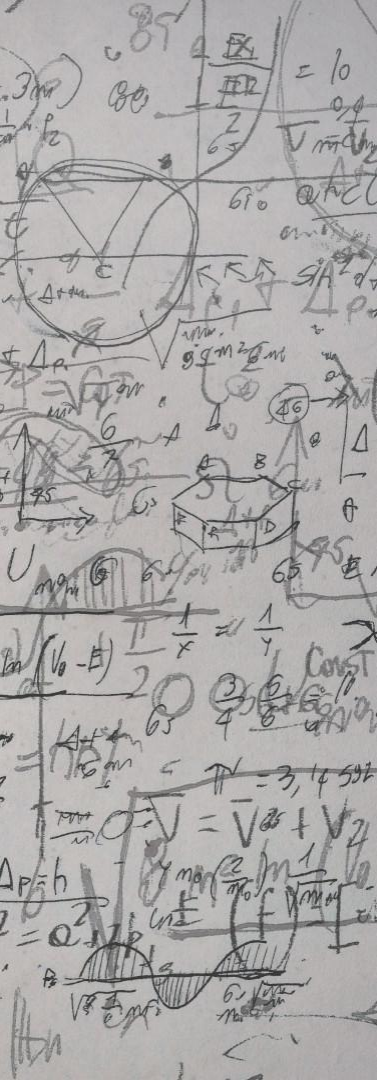
Des questions ?
Prenez-note ✏ :

Les problèmes



Avant : (ré-)initialisation en 13 ou 18 étapes

```
Truc createTruc(int x) {
  Truc truc = new Truc(); // Nulls uniquement transitoires ?
  truc.setStatus(VALID);
  initBidule(truc, x, false);
  validate(truc);
  if (someCondition()) {
    initBidule(truc, x, true); // Réinitialisation !
    truc.setStatus(INVALID); // Un objet à demi-initialisé est valide !
  }
}
```



Après : initialisation en une étape

```
Truc createTruc(int x) {
  Truc truc = new Truc(
    someCondition() ? INVALID : VALID, // Décision à un endroit
    createBidule(x, someCondition())
  ); // Truc() se valide lui-même
}
```

Avant : appel de setter ou builder oublié

```
Truc truc1 = new Truc();
truc1.setOjdsfg(51);
truc1.setGjdkfh(15);
truc1.setPbsugi(59);

Truc truc2 = Truc.builder()
        .pbsugi(36);
        .ojdsfg(14);
        .ybchjo(68);
        .build();
```

Avant : appel de setter ou builder oublié

```
Truc truc1 = new Truc();
truc1.setOjdsfg(51);
truc1.setGjdkfh(15); // Manquant dans truc2
truc1.setPbsugi(59);

Truc truc2 = Truc.builder()
        .pbsugi(36);
        .ojdsfg(14);
        .ybchjo(68); // Manquant dans truc1
        .build();
```

Après : erreur de compilation si paramètre oublié

```
Truc truc1 = new Truc(51, 59, 15, -1);

Truc truc2 = new Truc(14, 36, -2, 68);
```

Avant : dommages collatéraux

```
Bidule bidule = getBiduleById(id);
trucs.get(0).setBidule(bidule);
trucs.get(1).setBidule(bidule);

// Quelques méthodes plus tard...
trucs.get(1).getBidule().setHeight(42);
```

Après : aucun souci

```
Bidule bidule = getBiduleById(id);
trucs.get(0).setBidule(bidule);
trucs.get(1).setBidule(bidule);

// Quelques méthodes plus tard...
Truc truc = trucs.get(1);
truc.setBidule(truc.getBidule().withHeight(42));
```

Avant : multi-threading

```
public synchronized boolean wouldBeImportantWith(Truc truc) {
    setTruc(truc);
    recomputeState();
    return getState() == State.IMPORTANT;
}
```

Après : multi-threading

```
public boolean wouldBeImportantWith(Truc truc) {
    return withTruc(truc).getState() == State.IMPORTANT;
}
```


Une solution

Qu’est-ce qu’un Value-Object immutable ?

Objet non-modifiable après sa construction

Créé en une seule opération atomique

Aucun setter

Ses champs sont finaux

Objet à copier pour en obtenir des variantes

Repasser par un constructeur, directement ou indirectement

Objet toujours dans un état cohérent

Pas de transitions d’états pendant l’initialisation

Le constructeur s’assure de cette cohérence

Exemples de classes immutables en Java

String

BigDecimal

Instant, LocalDate,
ZonedDateTime, etc.

Exemples de classes immutables métier

Une adresse postale

Changer un champ revient à déménager

Argent

Montant + devise

Nom complet

Prénom + nom

Couleurs

Mixer deux couleurs produit une troisième couleur

Numéros de téléphone, adresses IP…

C’est quand même contraignant !

Oui et non…

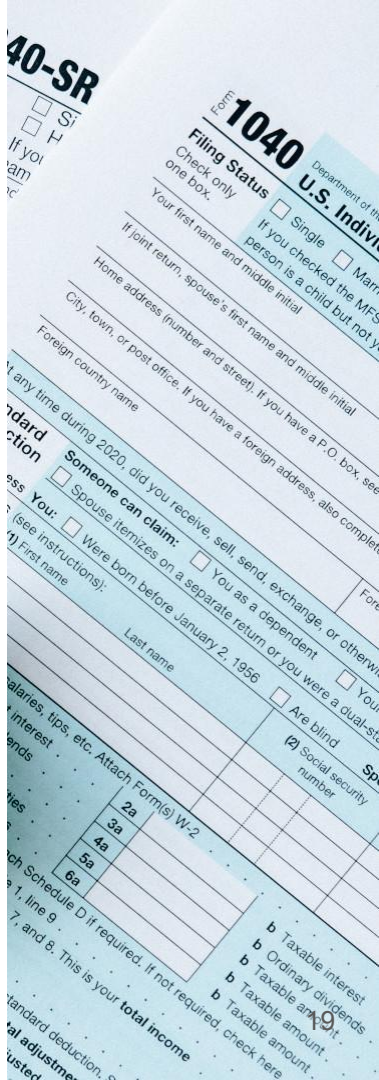
Chaque évolution ajoute des contraintes pour se faciliter la vie

Tests+boucles au lieu de GOTOs

Pas d’accès direct à la mémoire

Programmation orientée objet

Programmation fonctionnelle : immutabilité



Implémentation

Implémentation

Java :

```
public final class Color {
    private final String hex;

    public Color(String hex) {
        this.hex = hex;
    }

    public String getHex() { return hex; }
    // equals
    // hashCode
    // [toString] [compareTo]
}
```

Lombok :

```
@Value
public class Price {
    BigDecimal amount;
    Currency currency;
}
```

Java 16 :

(preview en 14 et 15)

```
public record FullName(
    String firstName,
    String lastName) {}
```

Attention : copier les objets mutables

```
List<String> names = new ArrayList<>(List.of("red", "green"));
Colors immutable = new Colors(names);
names.add("blue");
immutable.getNames().add("purple");
```

Attention : héritage fortement déconseillé

```
@Value class UserId { long id; }

class MutableUserId extends UserId {
    private long mutableId;
    public MutableUserId(long id) { super(id); mutableId = id; };
    public long getId() { return mutableId; }
    public void setId(long id) { mutableId = id; }
}
```

Autorisé : mutabilité en interne

```
class Lower {
    private /*final*/ String value;
    private /*final*/ boolean lazyLowered;
    public Lower(String value) { this.value = value; }
    public get() {
        if (!lazyLowered) {
            value = value.toLowerCase();
            lazyLowered = true;
        }
        return lazyLowered;
    }
}
```

Refactoring

L’application d’exemple

Les gestes métier

Insérer une ligne | Importer un fichier Excel

Produit	Vendeur	Prix	Devise	Conversion
CD Titanic	Auchan Villeneuve D'Ascq V2	14,99	EUR	14,99 EUR
CD Titanic	AliExpress.com	9,99	CNY	1,42 EUR

Chargement

Insérer une ligne

Changer un montant

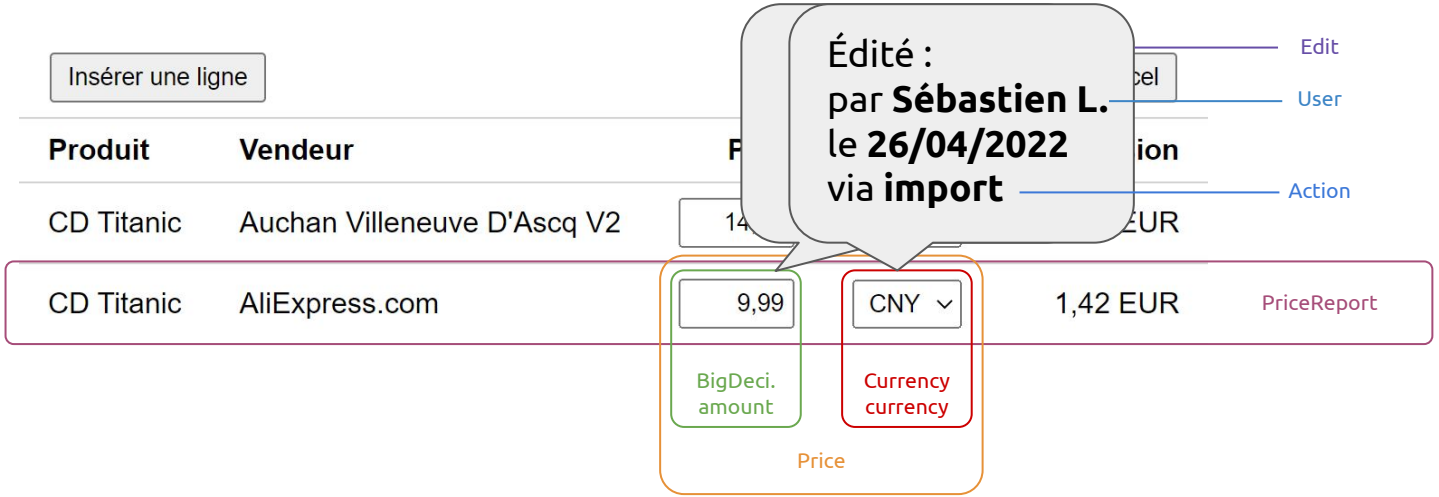
Changer la devise

Importer un fichier Excel

→ Éditions à historiser

Qui ? Quand ? Pourquoi ?

Les classes


Insérer une ligne
Produit
Vendeur
CD Titanic
Auchan Villeneuve D'Ascq V2
CD Titanic
AliExpress.com
9,99
CNY
1,42 EUR
Édité :
par Sébastien L.
le 26/04/2022
via import
Edit
User
Action
PriceReport
BigDeci.
amount
Currency
currency
Price

1/4 : préférez la composition à l'héritage

```java
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@NoArgsConstructor
@AllArgsConstructor
public class Edit extends User {

    private Instant instant;
    private Action action;
}
```

```java
@Data
@Builder

@NoArgsConstructor
@AllArgsConstructor
public class Edit {
    private User user;
    private Instant instant;
    private Action action;
}
```

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    String login;
    String fullName;
}
```

```java
@Value

public class User {
    String login;
    String fullName;
}
```

```java
creationEdit.setLogin(connectedUser.getLogin());
creationEdit.setFullName(connectedUser.getFullName());
```

```java
        creationEdit.setUser(connectedUser);
```

2a/4 : transformer en Value Objects immutables

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Edit {
    private User user;
    private Instant instant;
    private Action action;
}
```

```java
@Value
public class Edit {
    User user;
    Instant instant;
    Action action;
}
```

2a/4 : transformer en Value Objects immutables

```java
@Data
@AllArgsConstructor
public class Currency {
    private String code;
    private BigDecimal exchangeRateToEuro;
}
```

```java
@Value

public class Currency {
    String code;
    BigDecimal exchangeRateToEuro;
}
```

2a/4 : transformer en Value Objects immutables

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Price {
    private BigDecimal amount;
    private Edit lastAmountEdit;

    private Currency currency;
    private Edit lastCurrencyEdit;
}
```

```
@Value
public class Price {
    BigDecimal amount;
    Edit lastAmountEdit;

    Currency currency;
    Edit lastCurrencyEdit;
}
```

2a/4 : transformer en Value Objects immutables

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class PriceReport {
    private Product product;
    private Seller seller;
    private Price price;
}
```

```
@Data
@AllArgsConstructor
public class PriceReport {
    private final Product product;
    private final Seller seller;
    private Price price;
}
```

2b/4 : construire les objets par leurs constructeurs

GetPriceReportsUseCase

```
currency.setExchangeRateToEuro(upToDateExchangeRate);
report.setPrice(new Price(
        report.getPrice().getAmount(),
        report.getPrice().getLastAmountEdit(),
        new Currency(currency.getCode(), upToDateExchangeRate),
        report.getPrice().getLastCurrencyEdit()));
// TODO with*() or static factory
```

2b/4 : construire les objets par leurs constructeurs

InsertPriceReportRowUseCase

```
Edit creationEdit = new Edit();
creationEdit.setAction(Action.CREATION);
creationEdit.setInstant(Instant.now());
creationEdit.setUser(connectedUser);
```

```
Edit creationEdit = new Edit(connectedUser, Instant.now(), Action.CREATION);
```

```
Price price = Price.builder()
        // TODO[TRUE STORY] STRANGE: amount not set; intentionally
        // TODO[TRUE STORY] BUG: Missing amount CREATION Edit
        .currency(currencyRepository.getDefaultCurrency())
        .lastCurrencyEdit(creationEdit)
        .build();
```

```
Price price = new Price(
        null, // At creation, user has not entered any amount yet
        creationEdit,
        currencyRepository.getDefaultCurrency(),
        creationEdit);
```

```
PriceReport report = PriceReport.builder()
        .product(product)
        .seller(seller)
        .price(price)
        .build();
```

```
PriceReport report = new PriceReport(product, seller, price);
```

Tous les bugs étaient dans le vrai projet ayant servi d'inspiration

2b/4 : construire les objets par leurs constructeurs

UpdateAmountUseCase

```
report.getPrice().setAmount(newAmount);

Edit lastEdit = report.getPrice().getLastAmountEdit();
lastEdit.setAction(Action.EDITION);
// TODO[TRUE STORY] BUG: Missing Instant
lastEdit.setUser(connectedUser);
```

```
report.setPrice(new Price(
        newAmount,
        new Edit(connectedUser, Instant.now(), Action.EDITION),
        report.getPrice().getCurrency(),
        report.getPrice().getLastCurrencyEdit()));
// TODO with*() or static factory
```

2b/4 : construire les objets par leurs constructeurs

UpdateCurrencyUseCase

```
report.getPrice().setCurrency(newCurrency);

Edit lastEdit = new Edit();
lastEdit.setAction(Action.EDITION);
lastEdit.setInstant(Instant.now());
lastEdit.setUser(connectedUser);
report.getPrice().setLastAmountEdit(lastEdit); // TODO[TRUE
```

```
report.setPrice(new Price(
        report.getPrice().getAmount(),
        report.getPrice().getLastAmountEdit(),
        newCurrency,
        new Edit(connectedUser, Instant.now(), Action.EDITION)));
// TODO with*() or static factory
```

2b/4 : construire les objets par leurs constructeurs

XlsxImportUseCase

```
report.getPrice().setAmount(row.getNewAmount());

report.getPrice().setCurrency(newCurrency);

Edit lastEdit = report.getPrice().getLastAmountEdit();
// TODO[TRUE STORY] BUG: Missing Action.IMPORT
lastEdit.setInstant(Instant.now());
lastEdit.setUser(connectedUser);
// TODO[TRUE STORY] BUG: Missing lastCurrencyEdit
```

```
Edit lastEdit = new Edit(connectedUser, Instant.now(), Action.IMPORT);

report.setPrice(new Price(
        row.getNewAmount(),
        lastEdit,
        newCurrency,
        lastEdit));
```

3a/4 : créer des with*() & static-factories orientées métier

GetPriceReportsUseCase

```
report.setPrice(new Price(
        report.getPrice().getAmount(),
        report.getPrice().getLastAmountEdit(),
        new Currency(currency.getCode(), upToDateExchangeRate),
        report.getPrice().getLastCurrencyEdit()));
// TODO with*() or static factory
report.setPrice(report.getPrice().withSyncedExchangeRateToEuro(upToDateExchangeRate));
```

```
currency.setExchangeRateToEuro(upToDateExchangeRate);
report.setPrice(report.getPrice().withSyncedExchangeRateToEuro(upToDateExchangeRate));
```

3a/4 : créer des with*() & static-factories orientées métier

InsertPriceReportRowUseCase

```
Edit creationEdit = new Edit(connectedUser, Instant.now(), Action.CREATION);

Price price = new Price(
        null, // At creation, user has not entered any amount yet
        creationEdit,
        currencyRepository.getDefaultCurrency(),
        creationEdit);
```

```
Price price = Price.create(
        currencyRepository.getDefaultCurrency(),
        Edit.now(connectedUser, Action.CREATION));
```

3a/4 : créer des with*() & static-factories orientées métier

UpdateAmountUseCase

```
report.setPrice(new Price(
        newAmount,
        new Edit(connectedUser, Instant.now(), Action.EDITION),
        report.getPrice().getCurrency(),
        report.getPrice().getLastCurrencyEdit()));
// TODO with*() or static factory
```

```
report.setPrice(report.getPrice().withEditedAmount(
        newAmount,
        Edit.now(connectedUser, Action.EDITION)));
```

3a/4 : créer des with*() & static-factories orientées métier

UpdateCurrencyUseCase

```
report.setPrice(new Price(
        report.getPrice().getAmount(),
        report.getPrice().getLastAmountEdit(),
        newCurrency,
        new Edit(connectedUser, Instant.now(), Action.EDITION)));
// TODO with*() or static factory
```

```
report.setPrice(report.getPrice().withEditedCurrency(
        newCurrency,
        Edit.now(connectedUser, Action.EDITION)));
```

3b/4 : créer des with*() & static-factories orientées métier

```
public class Edit {
    User user;
    Instant instant;
    Action action;
}
```

```
public class Edit {
    User user;
    Instant instant;
    Action action;

    public static Edit now(User user, Action action) {
        return new Edit(user, Instant.now(), action);
    }
}
```

```
@Value
public class Currency {
    String code;

    BigDecimal exchangeRateToEuro;
}
```

```
@Value
public class Currency {
    String code;

    @With
    BigDecimal exchangeRateToEuro;
}
```

```java
public static Price create(Currency currency, Edit creationEdit) {
    return new Price(
            null, // At creation, user has not entered any amount yet
            creationEdit,
            currency,
            creationEdit);
}

public Price withEditedAmount(BigDecimal editedAmount, Edit edit) {
    return new Price(
            editedAmount,
            edit,
            this.currency,
            this.lastCurrencyEdit);
}

public Price withEditedCurrency(Currency editedCurrency, Edit edit) {
    return new Price(
            this.amount,
            this.lastAmountEdit,
            editedCurrency,
            edit);
}

public Price withSyncedExchangeRateToEuro(BigDecimal syncedExchangeRateToEuro) {
    return new Price(
            this.amount,
            this.lastAmountEdit,
            currency.withExchangeRateToEuro(syncedExchangeRateToEuro),
            this.lastCurrencyEdit);
}
```

4/4 : centraliser les règles métier structurantes

```
public Price(BigDecimal amount, Edit lastAmountEdit, Currency currency, Edit lastCurrencyEdit) {
    this.amount = amount; // Null if user did not enter anything yet, or if user erased the input
    this.lastAmountEdit = Objects.requireNonNull(lastAmountEdit, "lastAmountEdit");
    this.currency = Objects.requireNonNull(currency, "currency");
    this.lastCurrencyEdit = Objects.requireNonNull(lastCurrencyEdit, "lastCurrencyEdit");
}

public Price withEditedAmount(BigDecimal editedAmount, Edit edit) {
    if (Objects.equals(this.amount, editedAmount)) {
        return this;
    }

    return new Price(
            editedAmount,
            edit,
            this.currency,
            this.lastCurrencyEdit);
}

public Price withEditedCurrency(Currency editedCurrency, Edit edit) {
    if (Objects.equals(this.currency, editedCurrency)) {
        return this;
    }

    return new Price(
```

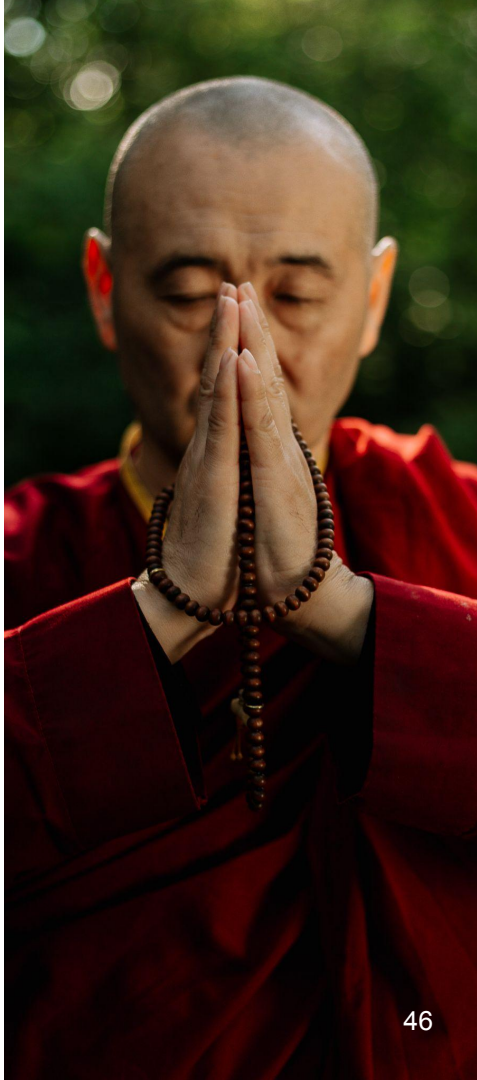
Gloire au compilateur !

Les bugs initiaux sont devenus des erreurs de compilation !

On a créé des méthodes réutilisables
décrivant les gestes métier

Le code est plus simple et descriptif
malgré les setX(getX().withY(y))

On vérifie l'intégrité dans le constructeur, passage obligé
même si on n'appelle quasiment jamais le constructeur directement




Mission accomplie
4 parties mobiles

Retrouver l'exercice

https://github.com/slaout/immutability-super-power-kata

Le kata

Branche "main"

La solution

Branche "java/exercise1/solution"

Une solution qui va plus loin

Branche "java/exercise1/solution-bonus"


Et après ?

Et Hibernate, dans tout ça ?

@Embeddable +
@Embedded +
@AttributeOverrides

EmbeddableInstantiator
Dans Hibernate 6
Sortie le 31 mars 2022

Niveaux suivants

Autres principes
Tell Don't Ask
Feature Envy
Loi de Déméter
Encapsulation

The Checker Framework : @NotNull / @Nullable

Domain Driven Design

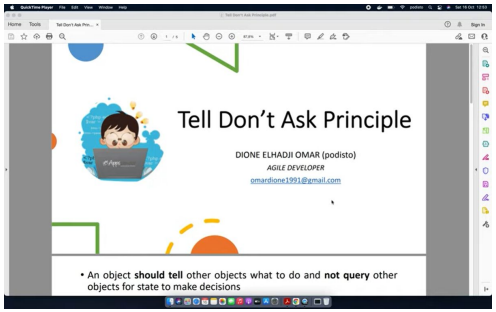
Programmation fonctionnelle

Vavr

Kotlin ?

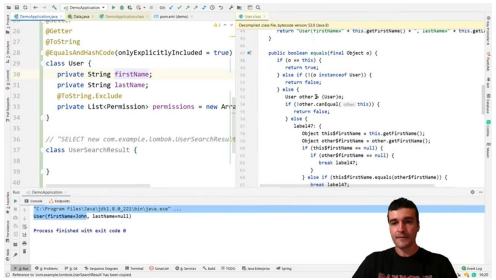
Liens

Vidéos



Tell Don't Ask Principle Kata

Le kata : https://kata-log.rocks/tell-dont-ask-kata
Explication : https://www.youtube.com/watch?v=36ILTQb_JpI
Solution partie 1 : https://www.youtube.com/watch?v=WKTdM6uObQQ
Solution partie 2 : https://www.youtube.com/watch?v=6cB0qUrTvQs



Decluttering Java - Project Lombok Best Practices

https://www.youtube.com/watch?v=DaOmyyRA8VU

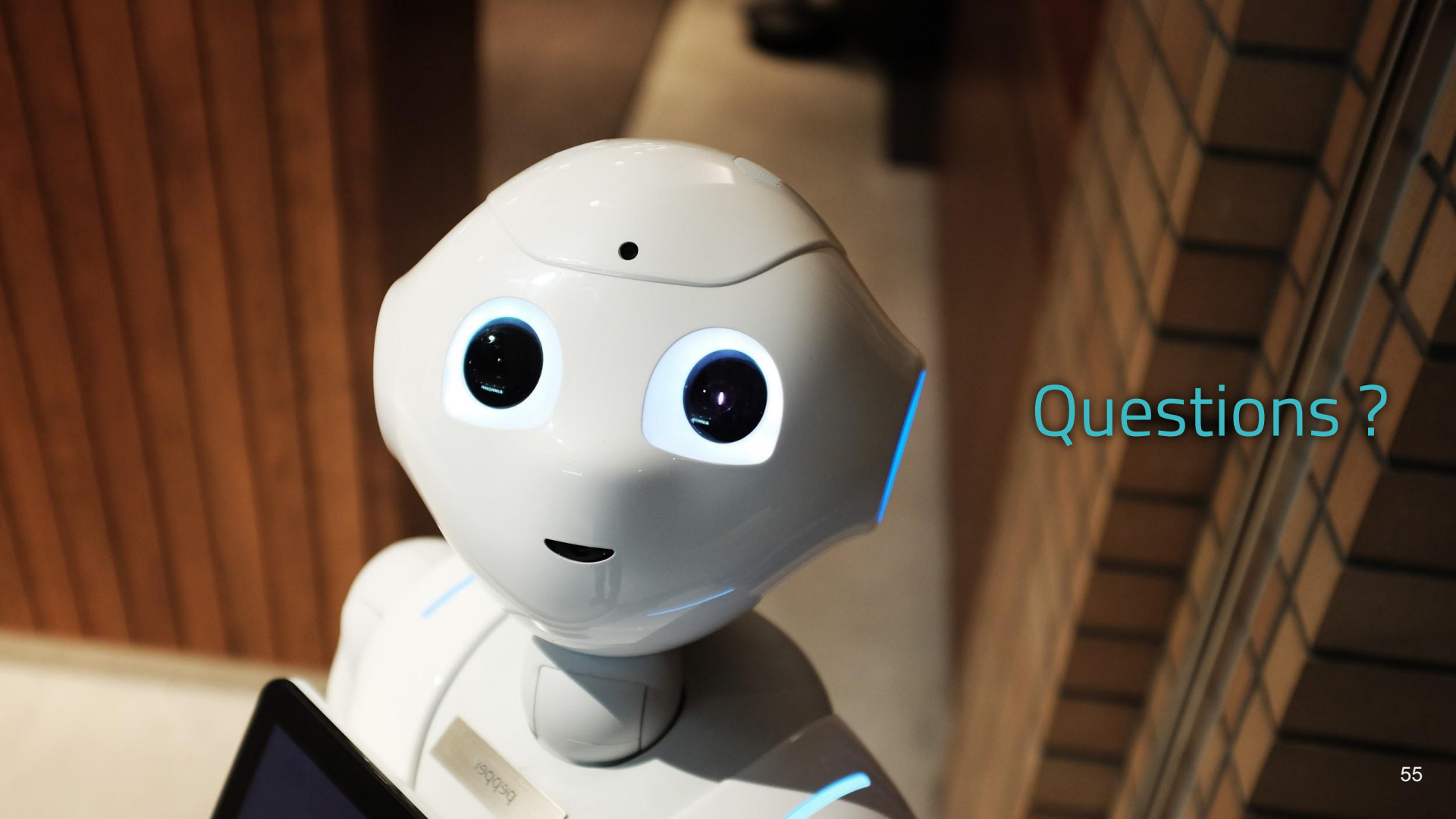
Aides mémoire de refactoring

(par Victor Rentea)

https://drive.google.com/drive/folders/1hvLAWeAgT753Mkb8zaHxNyBYwCewuWbb

IntelliJ

Eclipse


Questions ?

Crédits des images

Voir les commentaires présentateur des diapositives concernées sur https://docs.google.com/presentation/d/1T_q3zIphJUg_Vy9SHhShFILbqupDH3_8cYYvPaf4l5k