M3105 – TD : Une SOLIDe revue de code

*Ce TD sera réalisé en **mode déconnecté** :*
Laissez vos ordinateurs éteints pour le moment et prenez une feuille de papier 😉

Les bouts de code suivants sont extraits du chapitre **SOLID principles** du livre **What to Look for in a Code Review** de Trisha Gee disponible sur leanpub à l'adresse suivante : https://leanpub.com/whattolookforinacodereview

Pour chaque extrait de code suivant :
- Identifiez le principe SOLID non respecté
- Ecrire l'énoncé de ce principe (en le recherchant dans le cours)
- Dites en quoi ce principe n'est pas respecté et suggérez des pistes possibles de refactoring pour rendre ce code plus SOLID

❑ Bout de code n°1 :

Imagine, we have an abstract `Order` *with a number of subclasses –* `BookOrder`, `ElectronicsOrder` *and so on.*

The `placeOrder` *method could take a* `Warehouse` *(entrepôt) and could use this to change the inventory levels of the physical items in the warehouse.*

Now imagine we introduce the idea of electronic gift cards, which simply add balance to a wallet but do not require physical inventory.

If implemented as a `GiftCardOrder`*, the* `placeOrder` *method would not have to use the* `warehouse` *parameter.*

```
3  ↓  public abstract class Order {
4  ↓  ⊟    public void placeOrder(Warehouse warehouse) {
5              warehouse.itemSold(getItemId());
6              // do other order-related activities...
7          }
8
9  ↓       protected abstract int getItemId();
10     }
```

```
+ public class GiftCardOrder extends Order {
+     private BigDecimal price;
+     private Customer customer;
+
+     public GiftCardOrder(Customer customer, BigDecimal price) {
+         this.customer = customer;
+         this.price = price;
+     }
+
+     @Override
+     public void placeOrder(Warehouse warehouse) {
+         customer.addBalanceToWallet(price);
+     }
```

Principe non respecté :
Enoncé de ce principe :

Pourquoi ce principe n'est-il pas respecté ? :
(Utiliser le code de test donné en annexe 1 pour justifier ce point)

Vers un code plus SOLID… (suggérez des pistes possibles de refactoring) :

❑ Bout de code n°2 :

TweetMonitor.java @ e7595e3

```
public class TweetMonitor implements MessageListener<String> {

    private final Map<String, TwitterUser> allTwitterUsers = new HashMap<>();

    @Override
    public void onMessage(String twitterHandle, String fullTweetJson) {
        TwitterUser twitterUser = allTwitterUsers.computeIfAbsent(twitterHandle, Twit
        twitterUser.addMessage(getTweetMessageFromFullTweet(fullTweetJson));
    }

    private String getTweetMessageFromFullTweet(String fullTweetJson) {
        String textFieldName = "\"text\":\"";
        String nextFieldName = "\",\"source\":\"";
        int indexOfTextField = fullTweetJson.indexOf(textFieldName) + textFieldName.l
        int indexOfEndOfText = fullTweetJson.indexOf(nextFieldName);
        return fullTweetJson.substring(indexOfTextField, indexOfEndOfText);
    }
}
```

TweetMonitor.java @ 61b269e

```
public class TweetMonitor implements MessageListener<String>, Drawable {
    private static final int LEADERBOARD_ROWS = 10;
    private final Map<String, TwitterUser> allTwitterUsers = new HashMap<>();

    @Override
    public void onMessage(String twitterHandle, String fullTweetJson) {
        TwitterUser twitterUser = allTwitterUsers.computeIfAbsent(twitterHandle, Twit
        twitterUser.addMessage(getTweetMessageFromFullTweet(fullTweetJson));
    }

    private String getTweetMessageFromFullTweet(String fullTweetJson) {
        String textFieldName = "\"text\":\"";
        String nextFieldName = "\",\"source\":\"";
        int indexOfTextField = fullTweetJson.indexOf(textFieldName) + textFieldName.l
        int indexOfEndOfText = fullTweetJson.indexOf(nextFieldName);
        return fullTweetJson.substring(indexOfTextField, indexOfEndOfText);
    }

    @Override
    public void draw() {
        List<TwitterUser> topTweeters =
                allTwitterUsers.values().stream()
                        .sorted(comparingInt(TwitterUser::getNumberOfTweets).r
                        .limit(LEADERBOARD_ROWS)
                        .collect(toList());
        getController().setTopTweeters(topTweeters);
    }
}
```

Diff

"This side-by-side diff shows that a new piece of functionality has been added to TweetMonitor, the ability to draw the top ten Tweeters in a leaderboard on some sort of user interface."

Remarque : si vous ne connaissez pas l'instruction `computeIfAbsent`, vous pouvez consulter l'annexe 1 pour en savoir plus…Et rassurez-vous ce n'est pas cette instruction qui vous empêchera de répondre ou vous aiguillera vers la bonne réponse 😉

Principe non respecté :
Enoncé de ce principe :

Pourquoi ce principe n'est-il pas respecté ? :

Vers un code plus SOLID… (suggérez des pistes possibles de refactoring) :

❑ **Bout de code n°3 :**

```
14 +    public void handleEvent(Event event) {
15 +        if (event instanceof PreLoad) {
16 +            ei.preLoad(entity);
17 +        } else if (event instanceof PostLoad) {
18 +            ei.postLoad(entity);
19 +        } else if (event instanceof PrePersist) {
20 +            ei.prePersist(entity);
21 +        } else if (event instanceof PreSave) {
22 +            ei.preSave(entity);
23 +        } else if (event instanceof PostPersist) {
24 +            ei.postPersist(entity);
25 +        }
26 +    }
```

Principe non respecté :
Enoncé de ce principe :

Pourquoi ce principe n'est-il pas respecté ? :

Vers un code plus SOLID… (suggérez des pistes possibles de refactoring) :

❑ **Bout de code n°4 :**

```
6  + public class StringCodec implements SimpleCodec<String> {
7  +
8  +     @Override
9  +     public String decode(Reader reader) {
10 +         throw new UnsupportedOperationException("Should never have to decode a String object");
11 +     }
12 +
13 +     @Override
14 +     public void encode(final String value, Writer writer) {
15 +         writer.writeString(value);
16 +     }
17 + }
```

Principe non respecté :
Enoncé de ce principe :

Pourquoi ce principe n'est-il pas respecté ? :

Vers un code plus SOLID… (suggérez des pistes possibles de refactoring) :

❑ **Bout de code n°5 :**

```
6 12  public class CustomerService {
13 +      private static final String INSERT_CUSTOMER_STATEMENT = "INSERT INTO Customers"
14 +                                                            + "(id, first_name, last_name)"
15 +                                                            + "VALUES"
16 +                                                            + "( ?, ?, ? )";
17 +      private Database database;
18 +
19 +      public CustomerService(Database database) {
20 +          this.database = database;
21 +      }
22 +
23 +      public void addCustomer(int id, String firstName, String lastName) {
24 +          try (Connection connection = database.getConnection();
25 +               PreparedStatement statement = connection.prepareStatement(INSERT_CUSTOMER_STATEMENT)) {
26 +              statement.setInt(1, id);
27 +              statement.setString(2, firstName);
28 +              statement.setString(3, lastName);
29 +
30 +              statement.executeUpdate();
31 +          } catch (SQLException e) {
32 +              doDatabaseErrorHandling(e);
33 +          }
34 +      }
35 +
 7 36      public void placeOrder(Customer customer, Order order) {
 8 37          customer.incrementOrders();
 9 38          order.placeOrder(getWarehouse());
10 39      }
```

Principe non respecté :
Enoncé de ce principe :

Pourquoi ce principe n'est-il pas respecté ? :

Vers un code plus SOLID… (suggérez des pistes possibles de refactoring) :

Annexe 1 : Complément bout de code n° 1

```
18      @Test
19      public void shouldRemoveItemFromInventory() {
20          List<Order> allOrderTypes = asList(new BookOrder(ITEM_ID),
21                                             new ElectronicsOrder(ITEM_ID),
22                                             new GiftCardOrder(CUSTOMER, PRICE));
23
24          for (Order order : allOrderTypes) {
25              Warehouse warehouse = mock(Warehouse.class);
26              order.placeOrder(warehouse);
27
28              verify(warehouse).itemSold(ITEM_ID);
29          }
30      }
```

Annexe 2 : Aide à la compréhension du bout de code n°2
A propos de `computeifAbsent` …

La méthode `computeifAbsent` permet de simplifier la gestion des caches depuis Java 8 (moins de code et plus lisible 😉)

En JAVA 7 et antérieur

On applique l'algorithme décrit précédemment : ici, on recherche une commande par son id. Cela nécessite 8 lignes de code :

```
private static Commande getCommandeById(int id, HashMap hashMap) {
 if (hashMap.containsKey(id)) {
    return hashMap.get(id);
 } else {
    Commande commande = new Commande(id); //
    hashMap.put(id, commande);
    return commande;
 }
```

Récupération depuis la base de données

Ajout dans le hashMap

En JAVA 8: ComputeIfAbsent

Depuis JAVA 8, il est possible de faire la même chose en une seul ligne :

```
private static Commande getCommandeById2(int id, HashMap hashMap) {
   return hashMap.computeIfAbsent(id, x -> new Commande(x));
 }
```

La méthode computeIfAbsent est invoquée de la suivante : le premier paramètre est la clé pour rechercher l'élément dans la Map, le deuxième est la méthode à appliquer pour construire l'élément s'il n'existe pas. Ce fonctionnement se base sur l'écriture LAMBDA pour le fournisseur de l'objet recherché. On constate immédiatement la simplicité de l'utilisation des Lambda Expressions dans ce genre de cas !

Extrait : https://blog.axopen.com/2014/05/java-8-map-computeifabsent-gestion-caches/