# **Liste des points à relever sur la globalité du code**

* **Longueur du code**

*Repérer les morceaux de code qui semblent trop longs et qui gênent la lisibilité/compréhension*

* **Attention aux paramètres de méthode**

*Plus de trois paramètres requière une bonne justification et ne devrait pas être utilisé dans tous les cas[[1]](#footnote-1). Passer un boolean en paramètre est une très mauvaise pratique[[2]](#footnote-2).*

* **Degrés d’indentation**

*Pas plus de deux degrés d’indentation*

* *If(….){*

*If(…){*

*for(….){*

*}*

*}else{*

*}*

*}*

* **Nommage (1 lettre, abréviations…)**

*Repérer les noms de variables qui n’ont aucun sens*

* *int a =0 ; Contact exp ;*
* **Conventions**

*Camel Case, Eviter les rappels de contextes dans le nom des variables, Under scores entre le sould et le when*

* *List<Contact> contactList,*
* *List<Contact> contacts,*
* *ShouldReturn200WhenReceiveParameter()*
* *shouldReturn200\_whenReceiveParameter()*
* **Duplication de code et code inutile**

*Duplication totale ou partielle du code il faut penser à extraire ce code*

* **Responsabilité d’une méthode, classe, interface**

*Repérer si une entité porte trop de responsabilités*

* **Signification**

*Vérifier qu’une variable ou le nom d’une méthode a bien la bonne signification*

* *contact*
* *expectedContact*
* *getContact(String email)*
* *findContactByEmail(String email)*
* **Design (objets vs primitifs…)**

*Repérer s’il est préférable de transformer des types primitifs en objet ou à l’inverse préférer les types primitifs à un objet*

* *Integer customerAge = 3*
* *int customerAge=3 ;*
* *String customerName;*

*String customerAge;*

*String customerType;*

* *public class Customer{*

*String name ;*

*String age ;*

*String type ;*

*}*

* **Exploiter les spécificités du langage ou framework**

*Repérer les simplifications du code possibles grâce aux nouvelles versions dernières versions des frameworks ou langages*

*assertAll(), Optional.or(), stream()….*

# **Liste des points à relever spécifiques aux classes de test**

* **Vérifier que les tests (pour tous les cas) sont bien présents**

*S’il y a un if il faut penser à vérifier que les tests sont bien présents pour tous les cas du if*

* **Optimisation de la vitesse d’exécution (Dirties context)**

*Repérer la présence de dirties context qui impactent considérablement la vitesse d’exécution des tests, préférer le clean de routes ou des repository plutôt que de recharger le contexte avant chaque test*

* *@DirtiesContext(classMode = DirtiesContext.ClassMode.AFTER\_EACH\_TEST\_METHOD)*
* *repository.deleteAll(); route.reset();*
* **Ne pas utiliser les constantes dans les tests et préférer les données inscrites en dur**
* *assertThat(contact.getType()).isEqualTo(TOUCHPOINT\_TYPE)*
* *assertThat((contact.getType()).isEqualTo(’’TOUCHPOINT\_TYPE’) ;*
* **Assert sur objet plutôt que de champ à champ**

*Repérer s’il préférable de faire des asserts sur deux objets plutôt que de vérifier chaque champ*

* **Pertinence des asserts**

*Le test est-il pertinent ? L’assertion a-t-elle sa place dans ce test ?*

* *shouldSaveContact(){*

*//Arrange*

*…*

*//Act*

*…*

*//Assert*

* + *verify(contactRepository)*

*.save(expectedContact) ;*

* *verify(companyRepository)*

*.save(expectedCompany);*

*}*

* **Correspondance entre l’assertion et l’intention**

*Repérer si l’intention du test est bien en adéquation avec l’assertion, sinon revoir l’assertion ou le nom du test*

* *shouldSaveContact(){*

*//Arrange*

*…*

*//Act*

*…*

*//Assert*

* + *verify(contactRepository)*

*.save(expectedContact) ;*

*}*

* *shouldSaveContact(){*

*//Arrange*

*…*

*//Act*

*…*

*//Assert*

* *verify(companyRepository)*

*.save(expectedCompany);*

*}*

* **Redondance d’intention**

*Repérer les tests avec la même intention et qui testent donc la même chose*

1. Clean Code page 40 Function arguments first alinea [↑](#footnote-ref-1)
2. Clean Code page 41 Flag Arguments. [↑](#footnote-ref-2)