5 - BONNES PRATIQUES

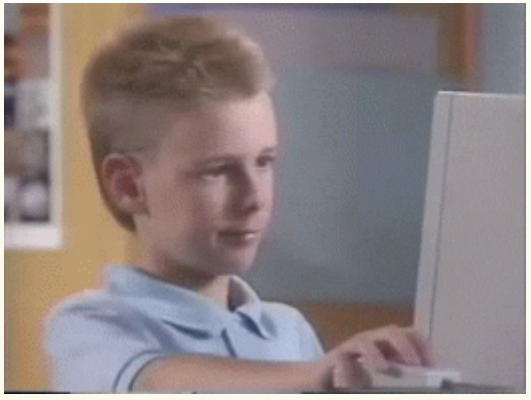
R4.02 : Qualité de développement

Institut Universitaire de Technologie de Bayonne et du Pays Basque

Pantxika Dagorret - Damien Urruty

BUT Informatique 2022 / 2023 - Semestre 4

BONNES PRATIQUES D'ÉCRITURES DE TESTS



PAR CONVENTION, LE NOM DE LA CLASSE DE TEST EST:

```
NomDeLaClasseTestée + Test(s)
```

UNE CONVENTION ALTERNATIVE:

```
NomDeLaClasseTestée + Should
```

PRIVILÉGIER LA NOTATION SNAKE_CASE PLUTÔT QUE LE CAMELCASE

```
@Test
void unCasDeTestAvecUnNomTresLongQuiNEstPasFacileALire() {
  // ...
}

@Test
void un_cas_de_test_avec_un_nom_tres_long_facile_a_lire() {
  // ...
}
```

ALLÉGER LE CODE EN UTILISANT LES IMPORTS STATIQUES

```
import static org.assertj.core.api.Assertions.assertThat;

@Test
void un_cas_de_test() {
  assertThat()...
}
```

RESPECTER LA STRUCTURE DU GIVEN WHEN THEN AVEC DES COMMENTAIRES

```
@Test
void devrait_etre_vainqueur_s_il_a_une_meilleure_attaque() {
    // GIVEN
    var premier = new Pokemon("Pikachu", "urlImage", new Stats(12
    var second = new Pokemon("Rondoudou", "urlImage2", new Stats(
    // WHEN
    var estVainqueur = premier.estVainqueurContre(second);
    // THEN
    assertThat(estVainqueur).isTrue();
}
```

RESPECTER LA STRUCTURE DU GIVEN WHEN THEN AVEC DES ESPACES

```
@Test
void devrait_etre_vainqueur_s_il_a_une_meilleure_attaque() {
  var premier = new Pokemon("Pikachu", "urlImage", new Stats(12
  var second = new Pokemon("Rondoudou", "urlImage2", new Stats(

  var estVainqueur = premier.estVainqueurContre(second);

  assertThat(estVainqueur).isTrue();
}
```

LE NOM DE LA MÉTHODE DE TEST DÉCRIT LE SCÉNARIO ET LE RÉSULTAT ATTENDU:

```
@Test
void devrait_afficher_une_erreur_si_meme_nom_de_pokemon_saisi() {
  // ...
}
```

Brenan Keller
@brenankeller

A QA engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 99999999999 beers. Orders a lizard. Orders -1 beers. Orders a ueicbksjdhd.

First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.

=> IL FAUT SE METTRE DANS LA PEAU DE L'UTILISATEUR !

S'ASSURER QUE LES MESSAGES D'ÉCHEC SOIENT INFORMATIFS

FAIRE ÉCHOUER LE TEST AVANT DE LE FAIRE PASSER

EVITER DE TESTER LES MÉTHODES PRIVÉES

TESTER CES MÉTHODES AU TRAVERS DES MÉTHODES PUBLIQUES

TESTER UN SEUL SCÉNARIO PAR TEST

EVITER LES ENCHAINEMENTS WHEN/THEN/WHEN/THEN...

NE PAS HÉSITER À SÉPARER EN PLUSIEURS TESTS

GARDER UNE LONGUEUR DE MÉTHODE DE TEST RAISONNABLE

MAX 10-15 LIGNES

SI LE TEST DEVIENT TROP LONG, CONSIDÉRER L'UTILISATION DE "BUILDERS"

```
var premier = unPokemon().avecAttaque(150).construire();
```

SI LE TEST DEVIENT TROP LONG, UTILISER DES MÉTHODES UTILITAIRES PRIVÉES

```
@Test
void devrait_etre_vainqueur_s_il_a_une_meilleure_attaque() {
  var premier = preparerPokemonAvecAttaque(150);
  var second = preparerPokemonAvecAttaque(149);

  var estVainqueur = premier.estVainqueurContre(second);

  assertThat(estVainqueur).isTrue();
}
```

PRÉFÉRER LA DUPLICATION PLUTÔT QUE LA FACTORISATION

AMÉLIORE LA LISIBILITÉ

PRÉFÉRER LA DUPLICATION PLUTÔT QUE LA FACTORISATION

```
@Test
void devrait_etre_vainqueur_s_il_a_une_meilleure_attaque() {
  var nomPokemon = "Pikachu";
  var attaquePokemon = 12;
  var defensePokemon = 9;
  var premier = new Pokemon(nomPokemon, attaquePokemon, defense
  var second = new Pokemon("Rondoudou", 11, 1);

  var vainqueur = bagarre.determinerVainqueur(premier, second);

  assertThat(vainqueur.getNom()).isEqualTo(nomPokemon);
  assertThat(vainqueur.getAttaque()).isEqualTo(attaquePokemon);
  assertThat(vainqueur.getDefense()).isEqualTo(defensePokemon);
}
```

PRÉFÉRER LA DUPLICATION PLUTÔT QUE LA FACTORISATION

```
@Test
void devrait_etre_vainqueur_s_il_a_une_meilleure_attaque() {
  var premier = new Pokemon("Pikachu", 12, 9);
  var second = new Pokemon("Rondoudou", 11, 1);

  var vainqueur = bagarre.determinerVainqueur(premier, second);

  assertThat(vainqueur.getNom()).isEqualTo("Pikachu");
  assertThat(vainqueur.getAttaque()).isEqualTo(12);
  assertThat(vainqueur.getDefense()).isEqualTo(9);
}
```

TESTS F.I.R.S.T.

BONNES PRATIQUES POUR L'ÉCRITURE DE TESTS

TESTS F.I.R.S.T.

RAPIDES *(FAST)*

- Des tests trop longs freinent le développeur
- Plus le nombre de tests est important, plus les tests doivent être rapides
- Une durée idéale est quelques millisecondes par test

TESTS F.I.R.S.T.

ISOLÉS / INDÉPENDANTS *(ISOLATED / INDEPENDENT)*

- Un test doit vérifier un et un seul comportement
- Pas de dépendance sur l'ordre d'exécution
- Pas de dépendance sur l'environnement
- Permet de lancer les tests en parallèle

TESTS F.I.R.S.T.

RÉPÉTABLES *(REPEATABLE)*

- Pas de dépendance sur l'environnement
- Comportement déterministe (pas de test dépendant du temps ou de l'aléatoire)

TESTS F.I.R.S.T.

AUTO-VALIDANT *(SELF-VALIDATING)*

- Pas besoin d'intervention manuelle
- Un test passe ou échoue, pas d'autre état possible

TESTS F.I.R.S.T.

EXHAUSTIFS / AU BON MOMENT *(THOROUGH / TIMELY)*

- Cas nominaux
- Cas alternatifs
- Cas d'erreur (mauvais paramètres, ...)
- De préférence écrits avant le code qui va avec

COMPLÉMENT: DESIDERATA DES TESTS

https://medium.com/@kentbeck_7670/test-desiderata-94150638a4b3

COMPLÉMENT: MODERN BEST PRACTICES FOR TESTING IN JAVA, PAR PHILIPP HAUER

https://phauer.com/2019/modern-best-practices-testing-java/