

LE TDD (Test Driven Development)

En quelques mots, le TDD est une technique de développement qui impose l'écriture de tests avant même l'écriture de la première ligne de code.

Nous allons trouver 5 étapes dans le TDD :

1. Ecrire un test
2. Vérifier qu'il échoue
3. Ecrire le code **suffisant** pour que le test passe
4. Vérifier que le test passe
5. Optimiser le code et vérifier qu'il n'y ait pas de régression (apparition d'un bug après refactorisation)

Nous pouvons englober ces étapes en 3 grandes idées :

- ❖ **Tester d'abord**, qui correspond aux deux premières étapes
- ❖ **Rendre fonctionnel**, qui correspond aux étapes 3 et 4
- ❖ **Rendre meilleur**, qui correspond à la dernière étape.

Pour représenter cela, nous allons prendre un exemple. Nous allons coder une fonction qui nous renvoie un booléen si l'année donnée en paramètre est bissextile.

Pour savoir si une année est bissextile il y a 4 critères :

- Si l'année est divisible par 400, l'année est bissextile.
- Si l'année est divisible par 100 mais pas par 400, l'année n'est pas bissextile.
- Si l'année est divisible par 4 mais pas par 100, l'année est bissextile.
- Si l'année n'est pas divisible par 4, l'année n'est pas bissextile.

Créons un nouveau projet Maven avec l'archétype Quickstart. Cela nous génère une solution avec une partie « main » où se situera notre code, et une partie « test » où nous mettrons nos tests unitaires.

Nous allons créer dans le dossier qui contient le fichier « App », une nouvelle classe que nous appellerons LeapYears. Dans cette classe, nous allons définir une nouvelle fonction que nous nommerons *isLeapYears*.

```
public class LeapYears {  
    public boolean isLeapYears(int year) {  
  
    }  
}
```

Nous la laisserons vide pour l'instant. Nous allons respecter le TDD et nous allons écrire le test pour le premier critère.

Pour écrire correctement un test, nous allons respecter une norme qui s'appelle AAA, pour Arrange, Act, Assert :

- La section Arrange d'une méthode de test initialise les objets et définit les valeurs des données qui vont être transmises à la méthode testée.
- La section Act d'une méthode de test appelle la fonction à tester avec les paramètres définis par la section Arrange.
- La section Assert vérifie que l'action de la fonction testée se comporte comme prévu.

Nous allons ajouter un test qui s'appelle

« `isLeapYearsYearDivisibleBy400ShouldReturnTrue` ». Le nom du test a l'air long et barbare, mais au moins nous avons une structure dans l'appellation :

- `isLeapYears` est le nom de la fonction qu'on teste.
- `YearDivisibleBy400` correspond à notre « arrange ».
- `ShouldReturnTrue` correspond au résultat attendu par la fonction, soit « l'assert ».

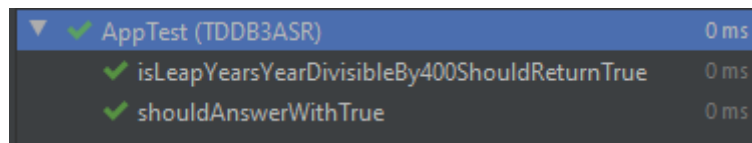
```
@Test
public void isLeapYearsYearDivisibleBy400ShouldReturnTrue() {
    int year = 2000; //Arrange -> Année divisible par 400
    LeapYears leapYears = new LeapYears(); //Constructeur de la classe LeapYears
    //Nous avons besoin d'une instance de la classe LeapYears afin de pouvoir ap-
    peler la fonction isLeapYears
    boolean isLeapYears = leapYears.isLeapYears(year); //Act -> On regarde le ré-
    sultat avec l'année que nous avons renseignée durant l'étape Arrange.
    Assert.assertTrue(isLeapYears); //Assert -> Le test est bon si isLeapYears est
    vrai.
}
```

Ici nous avons utilisé pour effectuer l'assert, la fonction `assertTrue()`. Cette fonction prend un paramètre un booléen et s'il est True, le test passera au vert. Il existe plusieurs fonction d'assert, nous en verrons d'autres avec les autres tests.

Maintenant, nous allons pouvoir coder notre fonction `isLeapYears` afin que le test passe et seulement ce test.

```
public boolean isLeapYears(int year) {
    if (year % 400 == 0) {
        return true;
    }
    return false;
}
```

Lançons ensuite les tests. (Clique droit dans AppTest -> Run AppTest). Tout doit être au vert !



▼ ✓ AppTest (TDDB3ASR)	0 ms
✓ isLeapYearsYearDivisibleBy400ShouldReturnTrue	0 ms
✓ shouldAnswerWithTrue	0 ms

Maintenant, nous allons écrire le test pour notre deuxième critère. Pour rappel, il s'agit de cela : *Si l'année est divisible par 100 mais pas par 400, l'année n'est pas bissextile.*

Nous allons créer le test suivant :

isLeapYearsYearDivisibleBy100ButNotBy400ShouldReturnFalse.

(La structure des tests reste identique, les commentaires auront donc disparu dans les prochains exemples).

```
@Test
public void isLeapYearsYearDivisibleBy100ButNotBy400ShouldReturnFalse() {
    //Arrange
    int year = 1900;
    LeapYears leapYears = new LeapYears();
    //Act
    boolean isLeapYears = leapYears.isLeapYears(year);
    //Assert
    Assert.assertFalse(isLeapYears);
}
```

Nous avons utilisé l'assert *assertFalse* pour vérifier que la fonction nous renvoie bien False. Le test passera si la valeur renvoyée par la fonction est False.

On lance les tests. Tout est au vert, nous n'avons pas besoin de corriger notre code. Passons au critère suivant.

Nous allons créer le test suivant :

isLeapYearsYearDivisibleBy4ButNotBy100ShouldReturnTrue.

```
@Test
public void isLeapYearsYearDivisibleBy4ButNotBy100ShouldReturnTrue() {
    //Arrange
    int year = 1960;
    LeapYears leapYears = new LeapYears();
    //Act
    boolean isLeapYears = leapYears.isLeapYears(year);
    //Assert
    Assert.assertTrue(isLeapYears);
}
```

Si nous lançons nos tests, nous voyons que le dernier que nous avons écrit ne passe pas. Il faut donc changer le code de notre fonction *isLeapYears*.

```

public boolean isLeapYears(int year) {
    if (year % 400 == 0) {
        return true;
    }
    if (year % 4 == 0 && year % 100 != 0) {
        return true;
    }
    return false;
}

```

Nous avons donc rajouté la condition si une année est divisible par 4 et pas par 100, alors elle est bissextile.

Nous relançons nos tests et tout passe !

Nous allons passer à notre dernier critère.

Le test est le suivant : *isLeapYearsYearNotDivisibleBy4ShouldReturnFalse*.

```

@Test
public void isLeapYearsYearNotDivisibleBy4ShouldReturnFalse(){
    //Arrange
    int year = 2019;
    LeapYears leapYears = new LeapYears();
    //Act
    boolean isLeapYears = leapYears.isLeapYears(year);
    //Assert
    Assert.assertFalse(isLeapYears);
}

```

Relançons les tests. Tout passe ! Notre code est donc correct et nous avons réussi à respecter le TDD !

Après cela il faudra donc optimiser le code et vérifier si les tests passent toujours ! (Ici nous n'allons pas le faire, car le code est déjà assez optimisé, mais nous pouvons écrire cette logique en une seule ligne de code).