TDD : Test-driven development

C’est une méthode qui permet de coder à partir de tests (Junit5 dans notre cas).

- Il faut donc dans un premier temps créer des test simples
- Ses tests ne passeront pas car rien n’a été codé.
- Nous codons les fonctionnalités dans le but de faire fonctionner les tests.

JMH : Benchmark (1 / 2)

C’est un outil de l’openJDK qui permet de mesurer les performances d’un code JAVA.
Pour exécuter les tests, il faut utiliser cette commande :

java –jar target/benchmarks.jar

```
LoggerBenchMark.faster_logger           avgt   15  0,867 ± 0,115  ns/op
LoggerBenchMark.faster_logger_disabled  avgt   15  0,931 ± 0,057  ns/op
LoggerBenchMark.no_op                   avgt   15  0,812 ± 0,214  ns/op
LoggerBenchMark.simple_logger           avgt   15  6,858 ± 1,354  ns/op
LoggerBenchMark.simple_logger_disabled  avgt   15  5,841 ± 1,364  ns/op
```

JMH : Les annotations (2 / 2)

Il existe de nombreuses annotations afin de configurer les tests :

- @Warmup(); permet de faire une phase de chauffe pour la JVM. Le code sera exécuté à blanc. Ceci permettra à la JVM d'allouer les ressources nécessaires pour que les tests suivants soient plus représentatifs.
- @OutputTimeUnit(); permet de définir l'unité de mesure pour nos tests.
- @Fork(); permet de définir le nombre de processus qui vont exécuter les tests.

Return Lambda VS Return Object

Avec les interfaces fonctionnelles, il est maintenant possible de retourner des lambdas. Lorsque l'occasion se propose, il beaucoup plus performant de retourner une lambda qu'un object.

```
LoggerBenchMark.faster_logger          avgt   15  0,867 ± 0,115  ns/op
```

- Comme nous pouvons le voir sur ces deux screens, le test ci-dessus qui retourne une lambda est beaucoup plus rapide que celui qui retourne un objet (le temps est en nanosecondes).

```
LoggerBenchMark.simple_logger          avgt   15  6,858 ± 1,354  ns/op
```

STATIC BLOCK

- En JAVA, il est possible de crée ce que l'on appelle des « static block ».

```
private static class C {
    private static final StringBuilder str = new StringBuilder();
    private static final Logger log = Logger.fastOf(C.class, msg -> str.append(msg));
    static {
        Logger.enable(C.class, false);
    }
}
```

- Le bout de code contenu dans le static{…} sera exécuter lorsque la classe C sera appelée pour la première fois (création d'objet ou appel d'une méthode static).
- Ce block peut nous permettre d'initialiser des objets statics ou par exemple d'effectuer une action la première fois que la classe est appelée.

MutableCallSite

- Les MutableCallSite sont des objets liés à des MethodHandle. Effectivement, un MethodHandle est stocké dans cet objet.

- On utilise un MutableCallSite lorsque le MethodHandle va se comporter comme un champs.

- Les deux principales méthodes sont :
  - getTarget(); retourne le MethodHandle lié au MutableCallSite
  - setTarget(); permet de changer le MethodHandle stocké.

MutableCallSite et les Threads

- La valeur récupérée par les threads n’est pas toujours la valeur mise à jour. Une mise en cache de l’ancienne valeur d’une MutableCallSite peut empêcher la mise à jour de la nouvelle valeur.

- Ce problème peut être contré en utilisant la méthode : MutableCallSite.*syncAll(MutableCallSite[]);* Elle a pour but de forcer le rafraichissement du cache afin de récupérer les valeurs mises à jour.