Développement dirigé par les tests

Julien Ponge – julien.ponge@insa-lyon.fr

Table of Contents

Projet Gradle

Gradle (http://gradle.org) est un outil de *build* de projet similaire à Maven, et qui utilise le langage Groovy (http://www.groovy-lang.org).

Décompresez l'archive `roman-numbers.zip` qui contient un squelette de projet Gradle configuré pour un projet Java.

Vous pouvez obtenir la liste des tâches disponibles en lançant `./gradlew tasks`.

1. Comment construire un projet ?
2. Comment générer de la *javadoc* ?
3. Comment lancer des tests ?

Gradle stocke ses fichiers dans `~/.gradle` de façon analogue à Maven qui utilise `~/.m2`.

JUnit et AssertJ

Soit la classe Java suivante qui ne produit rien de spectaculaire :

src/main/java/hello/Hello.java

```java
package hello;

public class Hello {

  public String hello(String who) {
    if (who == null) {
      throw new IllegalArgumentException("who cannot be null");
    }
    return "Hello " + who + "!";
  }
}
```

Quels sont les comportements *corrects* ?

Pour valider cette implémentation, nous écrivons nos *tests* avec JUnit (http://junit.org), et les *assertions* avec AssertJ (http://joel-costigliola.github.io/assertj/) :

src/test/java/hello/HelloTest.java

```java
package hello;

import org.junit.Test;
import org.junit.Before;

import static org.assertj.core.api.Assertions.*;

public class HelloTest {

  Hello hello;

  @Before
  public void setup() {
    hello = new Hello();
  }

  @Test
  public void check() {
    assertThat(hello.hello("Julien"))
      .isNotEmpty()
      .contains("Hello")
      .endsWith("!")
      .isEqualTo("Hello Julien!");
  }

  @Test
  public void check_null() {
    assertThatThrownBy(() -> hello.hello(null))
      .isInstanceOf(IllegalArgumentException.class)
      .hasMessageContaining("cannot be null");
  }
}
```

1. À l'aide de la documentation de JUnit et AssertJ, comprenez-vous ce test ?
2. Lancez les tests `./gradlew test`
3. Modifiez le code de la classe `Hello` pour faire échouer des tests. Lancez, comparez.
4. Gradle produit un rapport HTML de l'exécution de la suite de tests. Saurez-vous le trouver ?

JaCoCo

JaCoCo (http://www.eclemma.org/jacoco/) est un outil de *couverture de tests* : il permet de savoir quelles portions de code ont été testées (ou non).

Gradle propose un plugin JaCoCo (https://docs.gradle.org/current/userguide/jacoco_plugin.html) : intégrez-le dans votre build Gradle à l'aide de la documentation. Générez un rapport HTML de la couverture de tests, et observez un superbe 100% de couverture sur l'exemple fort complexe qu'est la classe `Hello`.

Nombres romains

Vous allez réaliser une classe `fr.insalyon.telecom.mgl.RomanNumberConverter` possédant les 2 méthodes suivantes :

- `int convert(String romanNumber)` qui converti un nombre romain en entier, et
- `String convert(int number)` qui converti un entier positif en nombre romain.

Pour rappel, les chiffres romains se constituent en assemblant :

Chiffre	**Valeur**	
I	1	
V	5	
X	10	
L	50	
C	100	
D	500	
M	1000	

On citera les règles suivantes depuis Wikipedia :

> La numérotation a été normalisée dans l'usage actuel et repose sur quatre principes :
>
> 1. Toute lettre placée à la droite d'une autre figurant une valeur supérieure ou égale à la sienne s'ajoute à celle-ci.
> 2. Toute lettre d'unité placée immédiatement à la gauche d'une lettre plus forte qu'elle, indique que le nombre qui lui correspond doit être retranché au nombre qui suit.
> 3. Les valeurs sont groupées en ordre décroissant, sauf pour les valeurs à retrancher selon la règle précédente.
> 4. La même lettre ne peut pas être employée quatre fois consécutivement sauf M.

Ainsi `VI` vaut 6, `IV` vaut 4, tandis que `VIIII` est invalide.

Implémentez cette classe en *itérations* successives.

1. Commencer par les cas simples (`X` , `I` , ...).
2. Écrivez *plein* de méthodes de test (`@Test`) plutôt qu'une géante.
3. Aidez-vous de JaCoCo pour garantir une bonne couverture de tests.
4. Affinez au fur et à mesure pour couvrir de plus en plus de cas.

> Une entrée incorrecte devra lever une `IllegalArgumentException` (`ABC` , `-10` , etc).

Pour en savoir plus sur les nombres romains :

- Numération romaine sur Wikipedia (http://fr.wikipedia.org/wiki/Numération_romaine)
- Un convertisseur en ligne (http://www.tinytools.nu/RomanNumeralsConverter/)

Intégration continue avec Jenkins

Vous allez apprendre à mettre en place un serveur d'intégration continue pour construire et tester automatiquement des projets logiciels.

Récupérez `jenkins.war` depuis http://jenkins-ci.org/ (http://jenkins-ci.org/). Installez-le dans `/tmp` ou autre endroit où vous avez de la place.

Jenkins utilise `~/.jenkins` et y stocke *beaucoup* de données.

Lancez Jenkins avec `java -jar jenkis.war` puis connectez vous à http://localhost:8080/.

Plugins (Chuck Norris inside)

Ajoutez les plugins suivants dans Jenkins :

- Git / GitHub
- JaCoCo
- Gradle
- Chuck Norris

Un premier build

Le code source de AssertJ est un projet Java, qui utilise Maven, et qui se trouve à https://github.com/joel-costigliola/assertj-core.

1. Créez un nouveau job.
2. Pointez sur le code source Git.
3. Dans les **triggers**, choisir **"poll SCM"** avec un **schedule** `@hourly`.
4. Dans la section build, le **root POM** est `pom.xml`, et les **goals and options** sont `clean install`.

Vérifiez que le build fonctionne. Corrigez au besoin votre configuration.

Modifiez la configuration et intégrez le plugin Chuck Norris. Relancez le build. Appréciez.

Nombres romains

1. Prenez votre projet de nombres romains et publiez-le sur GitHub.
2. Configurez un job Jenkins pour votre projet Gradle avec une périodicité de 2 minutes.
3. Incluez le support JaCoCo, sans oublier l'indispensable Chuck Norris.
4. Validez.

5. Modifiez votre projet pour introduire une erreur dans les tests. Publiez-la sur GitHub. Attendez un build. Savourez.
6. Corrigez. Publiez. Savourez.

Last updated 2015-06-08 13:26:30 CEST