POO : API & Outils
TD et TP

Anass OUSMOI anass.ousmoi@univ-lemans.fr
Florentin PAILLIER florentin.paillier@soprasteria.com

Table des matières

1. TP

1.1. Outillage de l'intégration continue (GitHub, Travis CI, CodeClimate, ReadTheDocs, Gitlab)

Nous allons utiliser la plate-forme d'hébergement de codes sources GitHub possédant un support impressionnant d'outillages pour tester/analyser/améliorer le code. Nous allons voir comment utiliser GitHub pour développer une plate-forme d'intégration continue.

Voici l'objectif du TP :

- Créer un projet Maven Spring Boot. Cela va créer les fichiers de configuration nécessaires pour un projet Spring. Nous pourrons ensuite initialiser un projet git pour commencer à suivre les modifications du projet. (1h)
- Une fois le projet créé et le projet Git initialisé, on va commencer à développer le programme Java en ajoutant des classes et des tests unitaires. (1h)
- Nous allons finir le TP avec la configuration de plusieurs outils d'analyses de code sources. (1h)

Vous obtiendrez à la fin plusieurs métriques

- Des résultats des tests unitaires avec Github Action,
- Une génération de la documentation de votre projet avec l'analyse de votre code source.

Etape 1

Commencez par générer votre application sur le site de Spring, avec Spring Initializr : https://start.spring.io/

Vous allez générer un projet Maven, avec Java, et Spring Boot 3.0.1 Dans le groupe, ajoutez le nom du groupe Maven du projet que vous souhaitez créer. Pour l'artifact, choisissez TP1.

Ajoutez la dépendance suivante :

- Spring Web

Générez votre projet :

Lancez votre IDE, configurez le proxy si nécessaire, et importez le nouveau projet (Vérifiez les aides plus bas dans le document si besoin).

Pour lancer une application Spring Boot, vous pouvez utiliser la commande maven suivante :

```
mvn spring - boot :run
```

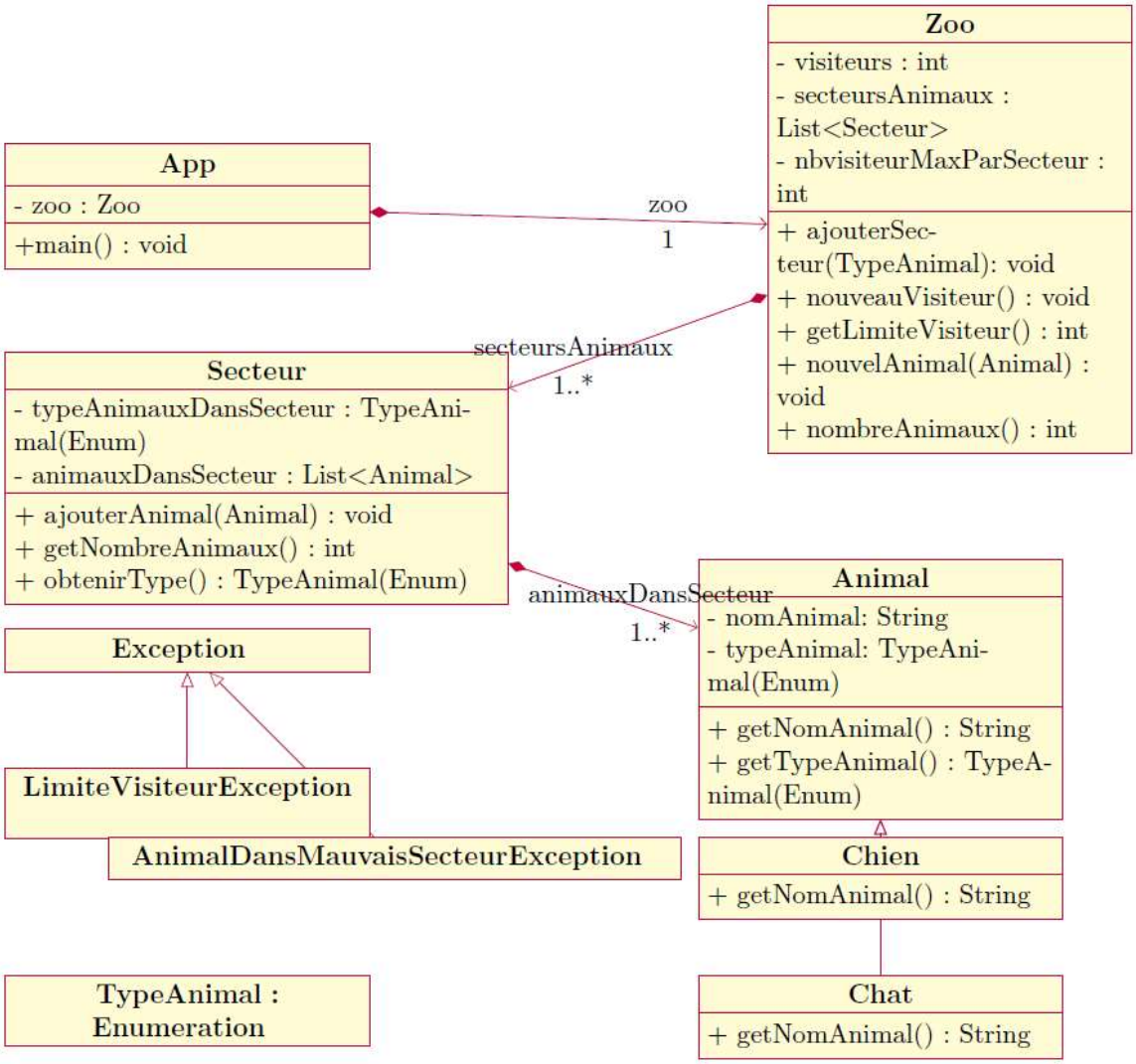
Créez une nouvelle « Run Configuration » de type Maven et dans le goal, utilisez la commande ci-dessus. Lancez la nouvelle configuration et regardez ce qui est inscrit dans le terminal pour voir si tout est OK.

Etape 2

Maintenant que le projet est créé, vous pouvez développer un petit programme Java.
Nous allons programmer une modélisation basique d'un zoo, voici ce que nous allons modéliser :

- Notre zoo contient des secteurs d'animaux, et quelques espèces d'animaux : chien, chat (par exemple).
- Ajouter des animaux au Zoo et les ranger dans leur secteur correspondant.
- Compter un nouveau visiteur dans le Zoo.
- Compter le nombre de visiteurs dans le Zoo.
- Avoir une exception si on dépasse le nombre de visiteurs dans le zoo: 15 visiteurs par secteurs maximum au sein du Zoo.

Voici la modélisation à développer :



Etape 3

Dans le fichier pom.xml de votre projet, utiliser la dépendance suivante pour les tests unitaires :

```xml
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

Etape 4
Créer le test unitaire suivant : si on dépasse le nombre maximum de visiteurs dans le zoo, une exception doit être lancée.

Etape 5
Créer le test unitaire suivant : si on ajoute un chien dans le zoo, il doit être stocké dans le bon secteur.

Etape 6
Lancez vos tests via Eclipse et via Maven avec la commande “mvn test”.

Etape 7
Créez ou utilisez votre compte GitHub pour créer un nouveau repository.

Etape 8
Créez un nouveau repository GitHub, et configurer par exemple GitHub Desktop/Vs-Code/Git ou en ligne de commande et ajoutez-y votre projet source.
Un proxy pour utiliser git en ligne de commande est parfois nécessaire (chaque encadré est une seule commande, ne pas sauter à la ligne après le “http.proxy”) :

```
git config -- global http.proxy http://proxy.univ-lemans.fr:3128
```

```
git config -- global https.proxy https://proxy.univ-lemans.fr:3128
```

Etape 9
Cherchez à quoi sert les Github Actions et votre projet pour les utilisez.

Etape 10
La page d’accueil d’un repository GitHub montre par défaut le fichier README.md.
Créez-en un s’il n’a pas déjà été rajouté par défaut : beaucoup d’exemples sont disponibles sur les autres projets opensource sur GitHub.

Regardez comment utiliser le badge du build status généré par votre Github Action et incorporez-le à votre README.md pour afficher un statut en temps réel de la validation de votre projet.
(Exemple : https://github.com/twbs/bootstrap , toutes les icones affichées dans le README.md sont des icones du même type que ceux fournis par Travis CI.)

Etape 11
Cherchez à quoi sert **codeClimate** et configurez-le pour fonctionner avec votre projet.

Etape 12
Ajoutez un nouveau test unitaire et validez son exécution sur votre github action.

Etape 13
Dans le fichier README.md de votre projet, ajoutez un lien pour pouvoir accéder au badge et pour accéder aux Github Actions et codeClimate.

Etape 14
(Info : GitHub et d'autres partenaires proposent plein d'outils professionnels exceptionnellement gratuit pour les étudiants : https://education.github.com/pack.)

Etape 15
Une fois tout configuré, envoyez-moi l'URL de votre projet GitHub par mail.

2. Astuces

2.1. Problèmes avec les dépendances Maven dans Eclipse

Voici quelques astuces pour vous débloquer (toutes indépendantes) :

1. Vérifier les options d'Eclipse pour s'assurer que le proxy est configuré et fonctionne (en essayant de voir la liste des plugins par exemple).
2. Maven dans Eclipse possède un fichier de configuration, visible ici : "Window>Préférences>Maven >User settings" (le chemin est prérempli s'il n'est pas modifié).
   Ajouter une partie pour configurer le proxy.
3. Configurez Maven en ligne de commande en téléchargeant la dernière version de Maven, ajoutez-le dans votre PATH (pour pouvoir l'utiliser en ligne de commande), et lancer la commande suivante dans le dossier de votre programme :

   ```
   mvn clean compile
   ```

   Le message d'erreur sera plus clair et pourra ˆetre plus facilement cherch´e sur internet.
4. Dans "Eclipse : clic droit sur le projet >Maven >Update project" pour prendre en compte les modifications du POM, et vérifier les d´dépendances.

2.2. Configuration du proxy dans les applications, Maven, Eclipse, Spring

2.2.1. Maven

Dans votre fichier de configuration Maven, utiliser ce fichier de configuration Maven pour le proxy à l'emplacement suivant "D:/Users/<login ENSIM>/.m2/settings.xml". Cas spécifique
Eclipse : Si le dossier .m2 comporte un dossier "repository" vide, supprimer dans un premier temps ce dossier.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<settings>
    <proxies>
        <proxy>
            <active/>
            <protocol>http</protocol>
            <port>3128</port>
            <host>proxy.univ-lemans.fr</host>
            <id/>
        </proxy>
    </proxies>
</settings>
```

Disponible au lien suivant : https://e-gitlab.univ-lemans.fr/snippets/1

2.2.2. Eclipse

"Windows >Préférences >General >Network Connections" avec les options suivantes pour le schéma HTTP, HTTPS et SOCKS (Et avec Active Provider `a Manual)
— Host : proxy.univ-lemans.fr
— Proxy : 3128

2.2.3. Programmation Java

```
java.net.Proxy proxyTest = new java.net.Proxy( java.net.Proxy.Type.HTTP , new
InetSocketAddress("proxy.univ-lemans .fr", 3128) );
OkHttpClient.Builder builder = new OkHttpClient.Builder(). proxy (proxyTest);
Starter.client = builder.build();
```

2.2.4. Spring

```
SimpleClientHttpRequestFactory clientHttpReq = new SimpleClientHttpRequestFactory();
Proxy proxy = new Proxy(Proxy.Type.HTTP , new InetSocketAddress ("proxy.univ-lemans.fr", 3128));
clientHttpReq.setProxy(proxy);
RestTemplaterestTemplate = new RestTemplate(clientHttpReq);
```

2.3. Problème de JDK/JRE avec Maven

Si Maven en ligne de commande ne fonctionne pas à cause d'un problème de JRE/JDK, allez vérifier la configuration JDK/JRE dans Eclipse dans les préférences : Windows, Préférences, Java >Installed JREs: il faut un JDK de configuré.

2.4. Problème de compilation non compatible JDK 5

La configuration par défaut de la version de Maven utilisé à l'ENSIM demande une rétrocompatibilité avec Java 5 sur tous les nouveaux projets via le plugin maven-compiler-plugin.
Pour modifier cette option dans vos projets Maven, vous pouvez ajouter ces paramètres dans le pom.xml qui va indiquer à la compilation d'être compatible avec le JDK 8 :

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>UTF-8</encoding>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Disponible au lien suivant : https://e-gitlab.univ-lemans.fr/snippets/11