POO : API & Outils
TD et TP

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

Karim.Rakhila@ soprasteria.com

Table des matières

TP 2 :

Objectifs pédagogiques

- Comprendre l'intérêt du modèle Inversion de contrôle. (Injection des dépendances)
- Apprendre à migrer une application Java vers Spring.
- Les Logs avec Log4j.
- Génération de Java doc
- Mise en place des APIs REST

Migration vers Spring

Etape 1 :

- Récupérer les sources de l'application « Zoo »
  - Source « A faire sous Github »
- Importer le projet dans votre environnement de développement via Maven.
- Lancer l'application App
- Intégration du BOM Spring

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                 <version>2.7.6</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

- Intégration Spring Context

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
```

Etape 2 (XML)

- Modifier la structure du projet de façon à avoir pour le model **Zoo** une Interface service et une Interface Dao
- Apprendre à migrer une application Java vers Spring.
    - Créer un fichier de configuration Spring (src/main/resources/application-config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xmlns="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd">


      <bean id="ZooDao" class="fr.ensim.tp.dao.ZooDao">
      </bean>

      <bean id="ZooService" class="fr.ensim.tp.service.ZooService">
                < constructor-arg ref="ZooDao"></constructor-arg>
Ou <property ref="ZooDao"></property>
      </bean>

</beans>
```

    - Création du contexte Spring

Le fichier étant dans le classpath du projet, nous utiliserons la classe Spring org.springframework.context.support.ClassPathXmlApplicationContext pour créer un contexte.

```
public static void main(String[] args) {
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("application-
config.xml");
}
```

    - Migrer l'application (modifier la méthode main())

Etape 3 (Annotation)

- Comprendre la configuration Spring basée sur Java
    - Créer une classe AppConfig
    - Configuration Spring

```
@Configuration
public class AppConfig {

}
```

    - Annoter les classes pour que l'application fonctionne (@Component, @Service , ..)
    - Créer une nouvelle classe « AppSpringJava » et lancer l'application avec new AnnotationConfigApplicationContext

Gestion de logs avec Log4j

Etape 4 :

Ajoutez la possibilité d'écrire des logs avec Log4j. Vous devez ajouter une nouvelle dépendance pour Maven, un fichier de configuration, et utiliser des méthodes pour écrire des logs.

- Ajoutez la dépendances suivantes dans le pom.xml de votre projet Maven

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

- Ajoutez une configuration pour écrire les messages de log dans la console et dans un fichier. Vous pouvez par exemple utiliser une configuration via fichier properties log4j.properties. (Exemple : chercher "log4j configuration" sur Google, mais attention à la version de la documentation).
- Utilisez le logger dans le code source
- Utilisez les différents niveaux de logs : trace, debug, info, warn, error, fatal. Regardez si vous voyez toutes les logs dans la console.

Génération de Java Doc.

Etape 6 :

- Ajouter la javadoc a vos métodes comme l'exemple ci-dessous :

```
/** * @param typeAnimal typeAnimal
 * @throws IllegalArgumentException
 */
public void ajouterSecteur(TypeAnimal typeAnimal) throws
IllegalArgumentException
```

- Executer la commande mvn javadoc:javadoc. Vous pouvez consulter la doc généré par maven dans le fichier target/site/apidocs/index.html
- On peut embarquer la génération de la doc automatiquement en ajoutant dans le pom le plugin ci-dessous dans la balise build.

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-javadoc-plugin</artifactId>
   <executions>
      <execution>
         <id>attach-javadocs</id>
         <goals>
            <goal>jar</goal>
         </goals>
      </execution>
   </executions>
</plugin>
```

- Lancer la commande mvn clean install et vérifier que vous avez bien la documentation sous forme de jar dans votre target.
- N'hésitez pas à rajouter un max de commentaires

Controller et appel rest

Etape 7 :

- Créer une classe ZooResource
- Dans ZooResource ajouter :

```
@RestController
@RequestMapping(path = "/zoo")
public class ZooResource {

    @Autowired
    IZooService zooService;

    static Logger LOGGER = Logger.getLogger(ZooResource.class);

    /**
     * Ressource permettant de compter le nombre de visiteur dans le zoo
     * @return le nombre de visteur du zoo
     */
    @GetMapping(path = "/visiteur", produces = "application/json")
    public int getNbVisiteur()  {
        LOGGER.info("Recherhe nombre visiteurs");
        return zooService.getNbVisiteur();
    }
```

- Dans Zoo service, ajouter l'annotation @Service et implémenter la méthode getNbVisiteur()
- Tester votre code en tapant localhost:8080/zoo/visiteur dans votre navigateur.

Astuces

1.1. 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.

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

1.2.1. 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

1.2.2. 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

1.2.3. 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();
```

1.2.4. 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);
```

1.3. 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.

1.4. 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