POO : API & Outils
TD et TP

Thibaut Smith
thibaut.smith@soprasteria.com
thibaut.smith-fisseau@univ-lemans.fr

2 novembre 2020

Les objectifs du cours (5 TD + 6 TP) :
- Prendre connaissance des nombreuses API Java
- Appréhender des concepts objets
- Utiliser un environnement de développement Java
- Savoir faire des tests unitaires
- Savoir utiliser les outils du développeur (maven, git, easymock)
- Un TP final noté devoir-maison
- Un DS
- Un TP noté (à confirmer)

Table des matières

1 TD

1.1 Utilisation d'Eclipse

Étape 1
Présentation du logiciel

Étape 2
Création d'un projet Java

Étape 3
Paramétrage des options

(a) Configuration éditeur

(b) Paramétrage plugins

(c) Paramétrage JDK

(d) Paramétrage Maven

Étape 4
Création d'un projet Maven

(a) Choisir "sans archetype"

(b) Ajouter une classe dans le package "src/main/java" contenant une méthode *public static void main(String[] args)* faisant un *System.out.println("Hello")*

(c) Vérifiez que votre projet se lance avec la commande "mvn clean compile".

Étape 5
Configurez la compilation du projet via votre IDE (Eclipse/IntelliJ, etc) et vérifier que vous arrivez à voir le message dans la console.

Étape 6
Développer une classe et l'instancier dans votre fonction *main*.

Étape 7
Configuration d'un debugger

Étape 8
Debug :

(a) Ajouts de points d'arrêts ("breakpoints") pour indiquer au debugger où s'arrêter.

(b) Vérifiez que le debugger s'arrête bien là où vous les avez posés.

Étape 9
Développement de classes et de tests unitaires

(a) Ajoutez un package *communication*

(b) Ajoutez-y une classe *Menu* permettant de demander quelques questions à l'utilisateur via *System.out*

(c) Récupérez les réponses via *System.in*

Étape 10
Vérifiez que le plugin Eclemma est déjà installé : il permet d'analyser la couverture de code.

Étape 11
Lancer la couverture de code.

1.2 Refactoring, utilisation Maven, et système de contrôle de versions

Étape 1

Principe du refactoring

Étape 2

Quelles parties du code pouvons-nous factoriser sur cet exemple ?

```
ArrayList<Integer> list1 = new ArrayList<Integer>();

list1.add(1);
list1.add(2);
list1.add(3);
list1.add(4);

ArrayList<Integer> list2 = new ArrayList<Integer>();

list2.add(5);
list2.add(6);
list2.add(7);
list2.add(8);

ArrayList<Integer> list3 = new ArrayList<Integer>();

list3.add(9);
list3.add(10);
list3.add(11);
list3.add(12);

int index = 0;
for(Integer value : list1) {
    System.out.println("list1.get(" + index++ + ") = "  +
       value);
}

index = 0;
for(Integer value : list2) {
    System.out.println("list2.get(" + index++ + ") = "  +
       value);
}

index = 0;
for(Integer value : list3) {
    System.out.println("list3.get(" + index++ + ") = "  +
       value);
}
```

https://e-gitlab.univ-lemans.fr/snippets/10

Étape 3

Tests unitaires et refactoring : TDD

Étape 4

Utilisation de Maven

(a) Cycle de vie des dépendances

(b) Commandes clean, compile, test, install, package, deploy

(c) Trouver les dépendances

Étape 5

Rappel des outils de versionnement de code source (SCM)

Étape 6

Stockez votre projet sur Gitlab ou GitHub

1.3 Modélisation UML

On souhaite développer une application permettant à deux personnes de jouer une partie d'échec en même temps sur le même ordinateur. Nous allons modéliser un programme permettant de jouer aux jeux d'échecs, utilisant ces règles-ci :

- Le jeu d'échecs se joue à deux joueurs qui font évoluer seize pièces chacun, respectivement blanches et noires, sur un échiquier de 64 cases en alternance blanches et noires (8 colonnes et 8 rangées).
- Jouer un coup consiste à effectuer un déplacement de l'une de ses pièces, accompagné éventuellement de la capture d'une pièce adverse se trouvant sur la case d'arrivée de la pièce jouée.
- Blancs et Noirs jouent à tour de rôle, par convention les Blancs jouent le premier coup de la partie.
- On ne peut pas passer son tour.
- Les bords de l'échiquier sont infranchissables par les pièces.
- Aucune pièce ne peut venir occuper une case déjà occupée par une pièce de son propre camp.
- Une pièce amie ou une pièce adverse qui se trouve sur la même colonne, rangée ou diagonale qu'une Dame, une Tour, ou un Fou constitue un rempart au-delà duquel ces pièces à longue portée ne peuvent sauter.
- Chaque joueur possède initialement : un roi, une dame, deux fous, deux cavaliers, deux tours et huit pions.

Étape 1

Faites le diagramme de classe de l'application.

(a) Quelles classes vont être nécessaires ?

(b) Chaque pièce possède un type de mouvement différent. Comment modéliser les particularités de chaque pièce ?

(c) Modéliser toutes les pièces :

1. Le roi
2. La reine
3. Les fous
4. Les tours
5. Les cavaliers
6. Les pions

(d) Modéliser la couleur de chaque joueur.

(e) Mises à part les classes liées aux règles du jeu, nous avons aussi besoin de représenter les deux joueurs, les pièces qui ont été capturées, et l'état de l'échiquier.

Étape 2

Comment faire pour vérifier la validité d'un mouvement fait par le joueur ?

Étape 3

Faites le diagramme de séquence du mouvement d'une pièce par un joueur. Note : une pièce peut être prise au cours du mouvement.

Étape 4

Nous souhaitons ajouter une fonctionnalité de sauvegarde de la partie pour pouvoir reprendre ultérieurement. Ajouter au modèle actuel les méthodes qui s'occuperont de la sérialisation/dé-sérialisation des objets.

[this umlcd style, anchor=north] (DeplacementPiece) at (-2,-19) **DeplacementPiece** second - debut : PositionPiece
- fin : PositionPiece ;
[this umlcd style, anchor=north] (PositionPiece) at (3.5,-19) **PositionPiece** second - colonne : Integer
- rangee : Integer ;
[this umlcd style, anchor=north] (Enum : CouleurJeu) at (9,-19) **Enum : CouleurJeu** second Blanc
Noir ;
[umlcd style, fill=purple, diamond-¿] (JeuEchec) – (Echiquier) node[near end, above]echiquier node[near end, below]1; [umlcd style, fill=purple, diamond-¿] (Echiquier) – (PiecesEnJeu) node[near end, above]pieces en jeu node[near end, below]1..*; [umlcd style, fill=purple, diamond-¿] (PiecesEnJeu) – (Piece) node[near end, above]pieces node[near end, below]1..*;

1.4 Questions d'examens des années précédentes

Étape 1

Examen 2018-2019 : écrire la surcharge du constructeur de la classe Voiture pour initialiser l'immatriculation en même temps que la voiture.

```
public class Voiture {
  private String immatriculation;
  public String getImmatriculation(){
    return this.immatriculation;
  }
  public void setImmatriculation(String immatriculation){
    this.immatriculation = immatriculation;
  }
  private String couleur;
  public String getCouleur(){
    return this.couleur;
  }
  public void setCouleur(String couleur){
    this.couleur = couleur;
  }
  public Voiture(String couleur) {
    this.setCouleur(couleur);
  }
}
```

Étape 2

Examen 2013-2014 : écrire une interface Comparable qui permet de comparer n'importe quelle classe ?

Étape 3

Examen 2013-2014 : À quoi servent les annotations en Java (donner plusieurs cas d'utilisations) ?

Étape 4

Ecrire une classe d'exception personnalisé NullArgumentException avec une méthode toString qui précise le nom de l'argument qui est null de la manière suivante : « L'argument X est null » avec le nom de l'argument X initialisé lors de l'instanciation de la classe.

Étape 5

Examen 2013-2014 : On désire modéliser des comptes en banque. Chaque compte est caractérisé par un numéro qui doit être unique et par le solde (la quantité d'argent sur le compte).

1. Définir une classe **Account** avec au moins deux constructeurs. Ajouter des méthodes *put* et *get* pour ajouter et retirer de l'argent. La méthode *get* doit faire en sorte que le solde ne devienne jamais négatif et retourne la somme qui a été effectivement retirée. Ajouter une méthode balance qui retourne le solde du compte.
2. Définir une nouvelle classe **Premium** pour un compte autorisant un découvert (solde négatif). Le découvert ne doit pas dépasser une valeur propre au compte qui peut être modifié par une méthode *decouvertAutorise*.
3. Définir une classe Bank qui peut contenir des comptes. Implémenter des méthodes *createAccount* et *createPremium* permettant respectivement de créer un compte et de créer un compte premium. Ces méthodes retournent l'identifiant du compte.

4. Écrire des méthodes *put* et *get* prenant en paramètre un numéro de compte et un montant et permettant d'ajouter et de retirer de l'argent. Comment faire pour que la méthode get prenne en compte le caractère premium du compte ?
5. Écrire une méthode *balance* qui donne le solde de la banque, c'est-à-dire, la somme des soldes de ses comptes.

Étape 6

Examen 2012-2013 : À quoi sert la méthode de Mocking (Mock) ?

2 TP

2.1 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 générique. Cela va créer les fichiers de configuration nécessaires pour un projet Java. 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 indicateurs de qualité de code avec *CodeClimate*,
— des résultats des tests unitaires avec *Travis CI*,
— une génération de la documentation de votre projet avec *Read The Docs*.

Étape 1

Vous allez commencer par créer un premier projet Java avec Maven. Utilisez Eclipse pour initialiser votre projet Maven :

— File, New, Maven Project,
— Choisissez "Create a simple project (skip archertype selection)",
— Remplissez la partie GAV du projet Maven (Group, Artifact, Version),
— Et cliquez sur "Finish" pour le créer.

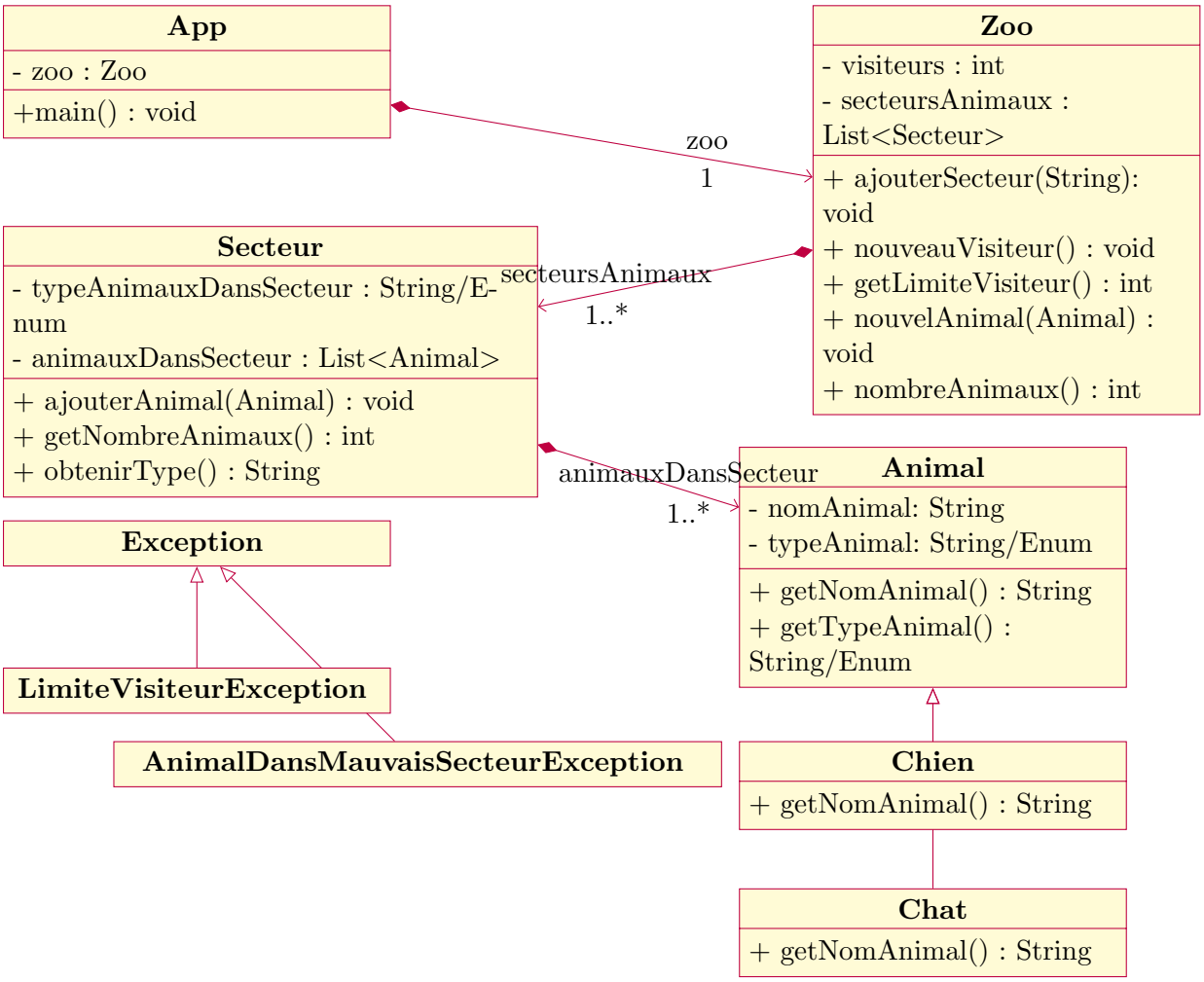
Étape 2

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 :



Étape 3

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

Étape 4

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

Étape 5

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

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

Étape 6

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

Étape 7

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

Étape 8

Créez un nouveau repository GitHub, et configurer **par exemple** GitKraken/Sublime-Merge/Git en ligne de commande 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
```

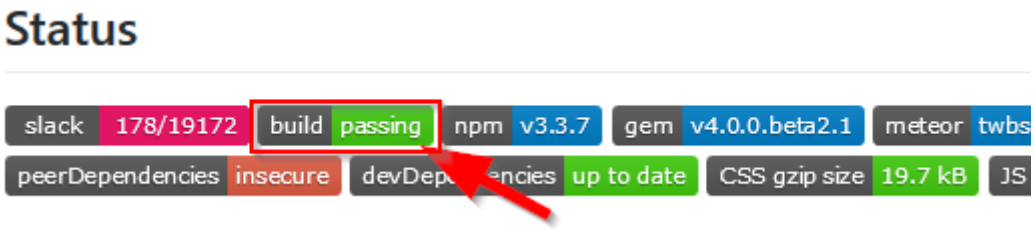
Étape 9

Cherchez à quoi sert **Travis CI** et configurez-le pour votre projet. Il a besoin d'un fichier de configuration *.travis.yml* pour analyser votre code.

Étape 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 **Travis CI** 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 icônes affichées dans le *README.md* sont des icônes du même type que ceux fournis par **Travis CI**.)



Étape 11

Cherchez à quoi sert **CodeClimate** et configurez-le pour fonctionner avec votre projet GitHub.

Étape 12

Créer le test unitaire suivant : si on ajoute un chien dans le zoo, le nombre d'animaux doit être correct. Y'a t'il un changement sur CodeClimate ?

Étape 13

Cherchez à quoi sert **ReadTheDocs** et configurez-le pour fonctionner avec votre projet.

Étape 14

Ajoutez un nouveau test unitaire et validez son exécution sur **Travis CI**.

Étape 15

Observez les indicateurs de **CodeClimate** et enlevez quelques *code smells* (= du code de mauvaise qualité entraînant une maintenance plus longue).

Étape 16

Allez voir la doc générée par **ReadTheDocs**, améliorez-la sur votre projet et observez la mise-à-jour sur **ReadTheDocs**.

Étape 17

Dans le fichier *README.md* de votre projet, ajoutez un lien pour pouvoir accéder au badge et pour accéder aux projets sur **Travis CI**, **CodeClimate**, et **ReadTheDocs**.

Étape 18

Une fois tout configuré, envoyez-moi l'URL (thibaut.smith@soprasteria.com) de votre projet GitHub par mail.

Étape 19

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

Étape 20

Prenez en compte les remarques de **CodeClimate** pour améliorer votre projet.

Étape 21

Bonus : Hébergez votre projet sous GitLab et trouvez un équivalent à Travis CI sur Gitlab, et essayez de le faire fonctionner.

- Créer un nouveau projet Gitlab
- Ajouter un nouveau 'remote' à votre projet Git
- Poussez vos modifications sur le projet Gitlab

3 Astuces

3.1 Problèmes avec les dépendances Maven dans Eclipse

Proxy + Maven + Eclipse = problèmes !
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 >Preferences >Maven >User settings" (le chemin est pré-rempli 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 être plus facilement cherché sur internet.
4. Dans "Eclipse : clique droit sur le projet >Maven >Update project" pour prendre en compte les modifications du POM, et vérifier les dépendances.

3.2 Configuration du proxy dans les applications, Maven, Eclipse, Spring

3.2.1 Maven

Dans votre fichier de configuration Maven, utiliser ce fichier de configuration Maven pour le proxy à l'emplacement suivant "/home/*user*/.m2/settings.xml", ou "D :/Profiles/*user*/.m2/settings.xml" (votre dossier personnel), ou "D :/Users/*login ENSIM*/.m2/settings.xml".

```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`

3.2.2 Eclipse

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

3.2.3 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();
```

3.2.4 Spring

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

        RestTemplate restTemplate = new
           RestTemplate(clientHttpReq);
```

3.3 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, Preferences, Java >Installed JREs : **il faut un JDK de configuré**.

3.4 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`