‡

Conception du Logiciel

TD n°1 : Jeux des personnages V1.0

‡ ‡ ‡

On se propose donc ici de concevoir et réaliser le début d'un petit jeu de personnages. Dans cette première version, nous modéliserons et implémenterons différents personnages, ainsi que la notion de troupe.

I - Le cahier des charges - V0.5

Plus précisément le jeu comporte pour le moment 3 personnages : l'ouvrier, le guerrier et le dragon. Un personnage humain appartient obligatoirement à une base définie par un nom et une couleur parmi rouge, jaune bleu ou vert.
Un ouvrier se défini par :

- un identifiant ; par défaut : ouvrier suivi du numéro de création de l'instance
- un niveau de vie (réel)
- une puissance de travail (entier)
- un ouvrier peut se déplacer en marchant

Un guerrier se défini comme ci-dessous :

- un identifiant ; par défaut : guerrier suivi du numéro de création de l'instance ;
- un niveau de vie (réel) ;
- une puissance d'attaque (entier), différente de la puissance de travail de l'ouvrier ;
- enfin, comme l'ouvrier, le guerrier peut marcher.

Concernant le dragon nous utiliserons la définition ci-dessous :

- un identifiant ; par défaut : dragon suivi du numéro de l'instance
- un niveau de vie (réel) ;
- une puissance d'attaque (entier), identique à celle du guerrier ;
- enfin un dragon peut se déplacer en volant.

Enfin, un guerrier n'étant là pour assurer la cueillette du muguet [1], celui-ci pourra attaquer [2] :

- un autre guerrier ;
- un ouvrier.

Exercice I.1

Conception UML - V0.5

Nous souhaitons concevoir une solution répondant à ce cahier des charges.

1. Proposer, à l'aide d'UMLDesigner, un diagramme de classe modélisant votre solution. On réfléchira en particulier à mettre en place les notions d'héritage et d'abstraction structurelle.

II - Implémentation - Maven & Eclipse

Nous souhaitons maintenant réaliser l'implémentation correspondant à la conception précédente.
Comme vous vous en apercevrez assez rapidement, la compilation, le build et l'exécution d'un projet Java peut vite s'avérer complexe. Aussi nous allons ici commencer à prendre en main un outils assez puissant permettant de faciliter et automatiser cette tâche : *Maven*

Exercice II.1

Création d'un projet maven

Avant tout commençons par créer les différents dossiers que vous utiliserez tout au long de ce cours.

1. Dans un terminal, naviguer vers la racine de votre dossier personnel.
2. Dans ce dossier, créer un répertoire Workspace-CLO.
   Ce répertoire sera la base de tous les développement que vous effectuerez en CLO.
3. Entrer dans le répertoire précédent et créer 1 nouveaux répertoire Projets
   C'est dans ce répertoire que vous créerez tous vos projets.

Avant de créer notre premier projet, une (très) brève présentation de l'outils *Maven*.

1. On considérera dans cette version que cette tâche (que nous n'implémenteront évidemment pas) est assurée par le dragon ...
2. Avec sa puissance d'attaque

Maven : présentation & création d'un projet

L'outils *Maven* est un outil proposé par *Apache* pour faciliter la création, la configuration, la gestion des dépendances et le cycle de vie (compilation, tests, build, versionning, etc.) d'un projet Java.
Lors de la création d'un projet à l'aide de maven, celui-ci va mettre en place un ensemble de répertoire permettant de structurer correctement votre projet, en particulier de séparer le code et les tests mais aussi des autres fichiers (image, documents, configuration par fichier xml, etc.). Plus précisément, vous trouverez à la racine du répertoire du projet créé :

- un répertoire src contenant 2 répertoires :
  - un répertoire main, qui contiendra toutes les sources, correctement organisées en package,
  - un répertoire test, qui contiendra, le moment venu, les fichiers de tests unitaires et de tests d'intégration.
- un répertoire target, après la première compilation uniquement, contenant le résultat de la compilation et/ou du packaging
- un fichier pom.xml, cœur du projet, puisque contenant toute la configuration de celui-ci.

Afin de pouvoir gérer correctement les dépendances avec d'autres éléments (autre de vos projets, ou projets externes), *maven* nécessite de formaliser le nommage des projets. Précisément, le nom d'un projet sera composé :

- d'un "groupId", tout en minuscule, séparé éventuellement par des points, et représentant l'institution auteur du projet ; par exemple : fr.ensma.a3.ia
- d'un 'artifactId", tout en minuscule, sans espace, correspondant au nom du projet (i.e. de l'artifact) que vous allez réaliser ; par exemple : test-java

Ces deux informations qui, in fine, identifieront complètement votre projet (fr.ensma.a3.ia.test-java) seront à fournir en paramètre dans la ligne de commande.
De plus, nous indiquerons également un "artifact modèle" afin de permettre la génération de la structure (i.e. des répertoires) et du squelette du fichier pom.xml)

4. Dans votre dossier Projets, saisir la commande ci-dessous afin de créer le projet nommé fr.ensma.a3.ia.jeux-personnage-v0.
```
mvn archetype:generate -DgroupId=fr.ensma.a3.ia -DartifactId=jeuxpersonnages-v0
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```
5. Parcourir l'arborescence des répertoires obtenus.
6. Que déduire de la position du fichier App.java ?

Nous allons maintenant compiler, puis construire ce nouveau projet jeux-personnage-v0. Voyons tout d'abord quelques commandes de bases de l'outils *maven*.

Maven : commandes de base

Cet encart, qui sera enrichi au fur et à mesure de l'avancement de ce cours, présentent différentes commandes de base de l'outil *maven*.

- mvn compile : compilation et construction des fichiers .class correspondants dans le répertoire target.
- mvn package : création d'une archive java, i.e. d'un fichier .jar, contenant les éléments du projet nécessaire à son utilisation dans un autre projet ou à son exécution. Ce fichier est également créer dans le répertoire target.
- mvn clean : nettoie le projet, c'est-à-dire supprime le répertoire target.

7. Compiler votre projet et remarquer la création du répertoire target.
8. Packager votre projet et remarquer l'apparition du fichier .jar.
   - Que déduire du nom donné par *maven* à ce projet ?
9. Nettoyer votre projet et remarquer la disparition du répertoire target.

Exercice II.2

Modification & exécution d'un projet maven

Afin de faciliter l'écriture de vos fichiers java, nous utiliserons l'IDE (integrated development environment) Eclipse. Voyons comment retrouver notre projet précédent au sein de cet IDE.

1. Lancer Eclipse et choisir comme répertoire de workspace le répertoire Workspace-CLO créer précédemment.

2. À l'aide du menu Fichier→import→ ouvrir la boite de dialogue d'import de projet.
3. Sélectionner maven→existing maven project, puis indiquer le répertoire contenant le fichier pom.xml du projet. Une fois terminée vous devez retrouver, dans la vue *package explorer*, la structure de votre projet (Cf. figure ci-dessous).
4. Dans le fichier App.java, modifier le message d'affichage de la fonction println.
5. Retourner dans le terminal et exécuter les commandes *maven* pour reconstruire le .jar.
6. Pour exécuter le projet, saisir la commande suivante :

```
java -jar target/jeupersonnages-v0-1.0-SNAPSHOT.jar
```

7. Expliquer l'erreur retournée par `java`.

Pour y remédier, nous devons modifier le fichier `pom.xml` afin d'y ajouter le chemin de la classe contenant la méthode main du projet.

8. Ajouter dans votre fichier pom.xml, après l'avoir ouvert dans Eclipse, la section comprise entre les balises `<build>` et `</build>` de manière à obtenir un fichier semblable à ci-dessous.

Listing 1 – : fichier pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>fr.ensma.a3.ia</groupId>
<artifactId>test-java</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>test-java</name>
<url>http://maven.apache.org</url>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <!-- Ajout du plugin pour créer un JAR exécutable -->
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>2.6</version>
      <configuration>
        <archive>
          <manifest>
            <mainClass>fr.ensma.a3.ia.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

9. Dans le terminal, reconstruire le fichier .jar
10. Exécuter le projet de nouveau et ... ça marche!!

Exercice II.3

Première implémentation du jeux de personnages

1. Au sein du projet créé précédemment , implémenter, sous Eclipse, votre conception précédente.
2. Ajouter dans la package racine, pour chacune des classes créées, une classe TestYYY[3] permettant d'instancier et de tester les fonctionnalités suivantes de ces classes :
   - `toString`
   - `equals` si nécessaire et `hashCode` dans ce cas
   - `clone`
3. Générer la documentation technique. Pour cela, commenter votre code à l'aide des commentaires de documentation et exécuter l'utilitaire mvn avec les bonnes options.

3. Où YYY représente le nom de la classe correspondante.

III - Le cahier des charges - V1.0

À l'aide de la notion d'interface logicielle, nous souhaitons ici modéliser le concept de troupe : plus précisément nous souhaitons regrouper l'ensemble des personnages pouvant se déplacer (i.e. l'ouvrier et le guerrier qui marche et le dragon qui vole).

Exercice III.1

Conception & implémentation de la troupe

1. Dans votre diagramme de classes UML :
   (a) Créer une interface *IDeplacable*.
   (b) Implémenter cette interface dans une/plusieurs classe(s) de votre arborescence.
   (c) Ajouter la classe *troupe*.
2. Implémenter, dans votre projet java, ces évolutions.
3. Tester.

IV - Tests & utilisation du cahier des charges - V1.0

L'implémentation étant maintenant terminée, nous passons à la phase de test [4].

Exercice IV.1

Tests unitaires et d'intégration

1. Ajouter, à l'aide du Framework *JUnit*, les tests unitaires des classes Ouvrier, Guerrier, Dragon et Troupe.
2. pour les méthodes le nécessitant, ajouter, à l'aide du Framework *JMockit*, les tests d'intégration.

Exercice IV.2

Valar morghulis ! !

Il est maintenant grand temps de se déplacer et de combattre ! !

1. Instancier un ouvrier, un guerrier et un dragon dans trois bases différentes.
2. Lancer une attaque, jusqu'à la mort, du guerrier sur l'ouvrier.
3. Instancier un nouvel ouvrier dans la base vide.
4. Former une troupe constituer des trois personnages et faire déplacer la troupe.

‡ ‡ ‡

4. Noter que cette phase aurait normalement due être réalisée avant l'implémentation pour les méthodes les plus complexes ...