TP2: jeu de rôle simplifié en Java

Introduction

Les objectifs de ce TP sont de commencer à se familiariser avec la notion de **classe** et d'**objet** (instances de classe). Dans cette optique, nous allons coder un petit jeu de rôle (*role-playing game* (RPG)) très simplifié. Ceci nous permettra notamment d'utiliser des *attributs* (variables d'instance) et des méthodes (d'instance).

(a) (b) (c)

Figure 1: Chevaliers en armure.

Spécifications

Dans ce RPG, il s'agira de définir des personnages qui possèderont certaines caractéristiques. Il semble donc logique de définir une classe "Character" ("Personnage" en français) avec des attributs et méthodes appropriés aux spécifications que nous souhaintons représenter.

On vous demande de respecter les critères suivants pour la classe Character:

- Un personnage a un nom.
- Un personnage a nombre points de vie (`HP`[1]) positifs, qui peut varier dans le temps, mais sans dépasser un maximum.

[1] "Health (or hit) Points" en anglais.

- Un nombre de points de vie à 0, entraîne la mort.
- Un personnage peut avoir une armure.
- Un personnage possède une certaine somme d'or (initialement 0).
- Un personnage peut être en train d'effectuer une quête (une seule à la fois pour simplifier).
- Un personnage peut être blessé ou soigné.
- Une blessure fait perdre un nombre de points de vie égal à ses dommages.
- La valeur de l'armure est retranchée des dommages.
- Des soins permettent de regagner un nombre de points de vie, sans dépasser le nombre initial.
- On ne peut soigner un personnage mort.

Dans notre RPG, les personnages pourront effectuer des quêtes[2] (quests en anglais). Une quête est une sorte de mission donné par un personnage spécial (personnage non-joueur (PNJ) ou non-player character (NPC)). Lorsque une quête est accomplie avec succès, une récompense est donnée au joueur par le PNJ donneur de quête.

Dans le cadre de ce TP, pour simplifier les choses, nous n'allons pas tenor compte des NPC. Nous allons simplement représenter les quêtes comme des instances d'une classe "Quest" qui aura un champ `description` (un `String`) et un champ `reward` (un `int`) et représentera une somme d'argent (nb de pièces d'or). Pour simplifier, les récompenses seront uniquement des pièces d'or et rien d'autre (pas d'arme ou d'élément d'équipement ou autres ...).

Remarque: un personnage et une quête seront dans une relation d'*aggrégation*. En effet, un personnage "a" une quête, autrement dit, une quête "appartient" à un personnage, mais les deux existent indépendemment l'un de l'autre. Par exemple, si un personnage meurt ou est "effacé" du jeu, une quête qu'il aurait eu dans son questlog (journal de quêtes) serait toujours existante dans l'absolu (deux joueurs *différents* peut effecteur la même quête en même temps, i.e., en parallèle).

Exercice 1

Nous vous proposons un diagramme de classe UML dans 2. Celui-ci représente la spécification présentée dans la section précédente.

Dans cet exercice, nous vous demandons de compléter les modificateurs de visibilité laissés exprès avec des "?".

[2]Le concept de quête existe par exemple dans https://en.wikipedia.org/wiki/World_of_Warcraft.

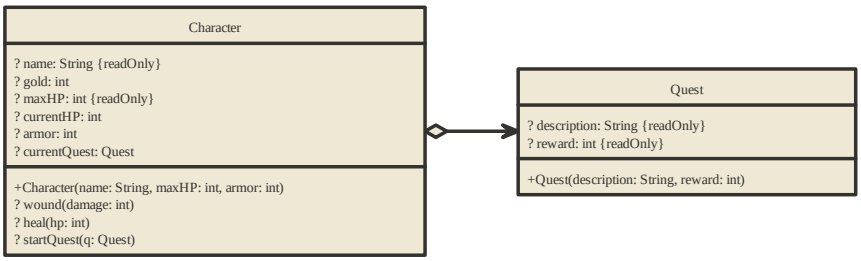

Figure 2: Diagramme UML incomplet.

Vous déposerez ce diagramme avec votre code source .java sur Moodle. Vous pouvez utiliser: https://nomnoml.com/ pour réaliser les diagrammes UML.

Exercice 2

1. Implémeter en Java les classes correspondantes au diagramme UML en tenant compte des modificateurs de visibilités que vous avez déterminés dans l'exercice 1.
2. Crééz aussi une classe "Jeu" qui va contenir la méthode main() pour exécuter votre programme avec quelque exemples d'instantiation de classes et appels de méthodes sur les objets. Vous pouvez par exemple vous inspirer de la Figure 1 et définir 3 chevaliers avec leur armure.
3. Implémentez aussi une méthode `toString()` pour chacune des classes. Cette méthode sera appellée dans le main() pour afficher textuellement les objets d'une meilleure manière que ce que fait l'implémentation par défaut.

La signature de la méthode est la suivante:

```
public String toString(){
  //votre implementation
}
```

L'idée est que nous voulons afficher textutllement l'*"état"* de l'objet, c'est à dire les valeurs de ses attributs, ce que ne fait pas l'implémentation par défaut.

Suggestion: Pour commencer, vous pouvez tester la méthode telle qu'implémentée par défaut et ensuite réfléchir à comment l'améliorer. N'hésitez pas à mettre en commentaire dans votre code votre compréhension par rapport à l'output de l'implémentation par défaut.

Packages

Un squelette avec une arborescence `ch.unige.cui.rpg` contenant des fichiers .java avec des déclarations de packages vous sont fournis sur Moodle. On importe le package `ch.unige.cui.rpg` dans notre main classe qui s'appelle Game.java.

Vous pouvez compiler directement tous les fichiers .java de ce package avec la commande suivante:

```
javac ./ch/unige/cui/rpg/*.java
```

Notez que le "." représente le dossier courant.

Finalement vous compilez Game.java et exécutez le programme, qui tel quel devrait simplement afficher "test1":

```
javac Game.java
java Game
```

Rendu

Ce TP n'est pas noté. Néanmoins, nous vous demandons de le déposer sur Moodle via le widget prévu à cet effet. Archivez votre code source et diagramme UML dans **un seul** fichier .zip avec la convention de nommage suivante: **prenomNomTP2.zip**.