Université Paris Cité **Programmation Orientée Objet**
L2 **Année 2024-2025**

TD - Séance n° 5

Héritage et Modélisation

On souhaite modéliser en Java les différents éléments d'un jeu de rôle très simplifié (personnages, actions de ces personnages).

1 Personnages et Actions

Modélisation des personnages. Chaque joueur choisit une classe de personnage parmi un éventail de possibilités, ici au nombre de quatre. Il existe deux grandes catégories de personnages : *guerrier* et *initié*. Un guerrier peut être soit un *paladin*, soit un *voleur*, tandis qu'un initié peut être soit un *sorcier*, soit un *moine*. Au début du jeu, chaque personnage reçoit un nom ainsi qu'un certain nombre de points de vie : 200 points pour un paladin, 150 pour un voleur, et 100 pour chaque type d'initié.

Exercice 1 *Hiérarchie des personnages*

1. Quelle est la hiérarchie naturelle permettant de représenter les personnages du jeu ? Quels seront les champs nécessaires, leur nature (statique ou d'instance) et leur visibilité (privée ou publique) ?
2. Quelles sont parmi ces classes celles dont on peut être sûr qu'on ne créera jamais d'instances (et donc, si vous connaissez cette notion, qui devraient être abstraites) ?
3. Donner une implémentation minimale des classes de la branche de cette hiérarchie permettant de créer un voleur.

Modélisation des actions. Lors de la création de son personnage, chaque joueur choisit deux actions possibles : une action primaire et une action secondaire, parmi deux grandes catégories d'actions : utiliser une arme ou lancer un sort. Une arme peut être soit une épée, soit un gourdin. Un sort peut être un sort de lumière, un sort des ténèbres ou un sort élémentaire.

Les effets des différentes actions sont similaires : ils entraînent une altération des points de vie de l'auteur de l'action (négative, positive ou nulle) et une altération des points de vie de la cible de l'action (idem). Seules les valeurs numériques de ces altérations varient d'une action à l'autre, selon les règles suivantes :

élément	effet sur l'auteur	effet sur la cible
épée	-10 pv	-30 pv
gourdin	+5 pv	-10 pv
sort des ténèbres	-5 pv	-20 pv
sort élémentaire	+5 pv	-10 pv
sort de lumière	aucun	+20 pv

Exercice 2 *Hiérarchie des actions*

1. Quelle est la hiérarchie naturelle permettant de représenter les actions ? Quels seront les champs et méthodes nécessaires dans les classes de cette hiérarchie ?
2. Donner une implémentation minimale des classes de la branche de cette hiérarchie permettant de créer une épée.
3. Quelles seront les modifications des classes de personnages nécessaires pour munir chaque personnage de deux actions dès sa création, et pour lui permettre d'effectuer ses actions primaire et secondaire sur une cible ?

Choix des actions. Les choix d'actions primaire et secondaire possibles pour un personnage sont déterminés par sa classe, selon les règles suivantes :

personnage	élément pour l'action primaire	élément pour l'action secondaire
Paladin	épée	gourdin ou sort élémentaire
Voleur	épée ou gourdin	sort des ténèbres
Sorcier	sort des ténèbres	sort de lumière ou élémentaire
Moine	gourdin	sort de lumière

Exercice 3 *Contraintes de construction*

1. Comment faire en sorte d'interdire la compilation d'un programme dans lequel on créerait explicitement un personnage non conforme aux règles du jeu ?
2. Donner une nouvelle version de la classe du voleur ne permettant de créer un voleur qu'en respectant cette contrainte.

Exercice 4 *Notions d'égalité de personnages et d'actions*

On souhaite redéfinir la méthode `equals()` de certaines des classes ci-dessus de manière à définir pour les actions et personnages les notions d'égalité suivantes :

— Deux actions seront considérées comme égales si et seulement si elles sont des instances de la même classe [1].

1. Dans cette modélisation simpliste, toutes les instances d'une même classe d'actions se valent : il est par exemple inutile de créer plus d'une seule instance d'une épée. Sans chercher à raffiner ce modèle, on pourrait éventuellement se contenter ici d'une égalité par référence.

— Deux personnages seront considérés comme égaux s'ils sont des instances de la même classe, s'ils portent le même nom, ont le même nombre de points de vie et des choix d'action primaire et secondaires égaux au sens de la notion précédente.

Dans quelles classes faut-il redéfinir `equals()`, et avec quelle implémentation ?

2 États du jeu

L'état courant du jeu sera représenté par une classe encapsulant la liste des personnages encore présents dans le jeu à un instant donné.

Exercice 5 *Représentation d'un état*

1. Définir une classe permettant de représenter l'état courant du jeu.
2. Comment proposeriez-vous d'implémenter la méthode `equals()` de cette classe ?

Exercice 6 *Clonage d'un état*

Afin de pouvoir conserver l'historique des états du jeu, on souhaite pouvoir faire une copie profonde d'un état (un clone).

1. En dehors de la classe représentant un état, dans quelles classes faudra-t-il réimplémenter (avec la visibilité `public`) la méthode `clone`, et comment ?
2. Donner l'implémentation de `clone` dans la classe représentant un état du jeu.

3 Bonus

Exercice 7 Dans une modélisation un peu plus élaborée du jeu, certains événements pourraient avoir un effet plus durable que la simple modification ponctuelle des points de vie des joueurs par leurs actions. Par exemple, un personnage pourrait être à chaque instant dans l'un des trois états suivants, et passer d'un état à l'autre en fonction du déroulement du jeu :

1. Normal : l'effet des actions du personnage est celui décrit en début d'énoncé.
2. Berserk : les altérations de chaque action sont multipliées par 2.
3. Fatigué : les altérations de chaque action sont divisées par 2.

Sans se préoccuper de notion d'égalité ou de clonage, comment modéliseriez-vous cet aspect du jeu en exploitant seulement l'héritage et la liaison dynamique, en particulier sans aucun `if` ?