Université Paris Diderot **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques **Année 2015-2016**

TP n° 5
vspace0.5cm Jouons

Dans ce TP, nous allons clore notre petite escapade médiévale en créant un mini-jeu sur le sujet. **Attention :** nous allons définir plusieurs classes relativement indépendantes avant de finalement les assembler à la fin, **pensez à faire des tests au fur et à mesure du TP**, n'attendez pas la fin.

Principe du jeu Nous allons repartir de la modélisation d'une société médiévale afin de créer un jeu simple. L'idée est de faire évoluer un royaume, constitué de villages, dans le but de le faire prospérer au maximum. Comme dans l'épisode précédent, un villages est composé de roturiers, sur lesquels un impôt (dont le montant correspond à la moitié de leur gain) est prélevable.

Le joueur sera en charge du royaume, et aura à chaque tour la possibilité d'effectuer des actions en fonction de l'argent qu'il a de disponible. Dans un premier temps, on considère qu'il peut :

— « acheter » un nouveau paysan, avec un certain coût ;
— construire un nouveau village, avec un certain coût aussi ;
— prélever l'impôt dans le royaume, ce qui correspond à le prélever dans chacun des villages ;
— finir le tour, qui a pour effet de faire passer une année, vieillir les personnages et effectuer une production.

Exercice 1 *Le royaume*

Les fichiers `Personne.java`, `Roturier.java`,`Paysan.java` vous sont fournis sur Didel. Vous constaterez que cette fois-ci, les roturiers produiront d'autant plus qu'ils ont de l'argent, et on n'aura donc pas intérêt à trop prélever systématiquement d'impôt.

1. Implémenter les classes `Village` et `Royaume`, qui contiendront des méthodes `vieillir, production, impot`. On veillera à faire mourir les villageois lorsque nécessaire.
2. Ajouter à la classe royaume une méthode `ajouteHabitant(Roturier r)` qui ajoute un habitant à l'un des villages du royaume, choisi au hasard ; ainsi qu'une méthode `depense`.
3. Ajouter aussi une méthode `toString()` afin de pouvoir faire un affichage du royaume.
4. Afin de préparer le terrain pour la suite, implémenter une classe `Fabrique` qui permette de créer des paysans (dont les caractéristiques initiales `capacité` et `pdv` sont les mêmes pour tous, par exemple 20 et 40) et des villages.

Exercice 2 *L'action*

Afin de représenter les différentes actions, nous allons utiliser une classe abstraite `Action`, dont hériteront différentes classes.

1. Créer une classe abstraite `Action`, en sachant que toutes les actions ont en commun le fait d'avoir un coût, un nom, ainsi qu'une méthode `void action(Royaume r, Fabrique f)` qui agit sur le royaume `r`, en utilisant au besoin la fabrique `f`. On cherchera à factoriser au maximum le code des actions, c'est-à-dire à en mettre le maximum dans la classe abstraite mère `Action`.
2. Écrire les classes `AchatPaysan` (coût 50), `Impot` et `FinTour` qui héritent de `Action`.
3. Écrire une classe `FabriqueAction`, qui contiendra :
   — un attribut `List<Action> liste`, qui contiendra une instance de chaque action possible ;
   — une méthode `boolean actionCorrecte(String nom)` qui renvoie `true` si `nom` correspond au nom d'une des actions ;
   — une méthode `Action creer(String nom)` qui renvoie l'action correspondant au `nom`.

Exercice 3 *Le jeu*

Il ne nous reste plus qu'à définir le jeu et les joueurs afin de pouvoir commencer le jeu.

1. Définir une interface joueur qui contiendra une méthode `public Action prochaineAction(int argent)`, par laquelle le joueur renverra la prochaine action qu'il souhaite effectuer, en sachant qu'il a `argent` écus disponible dans les caisses du royaume.
2. Définir la classe `JoueurInteractif` qui utilise un scanner pour demander la prochaine action à faire. On se servira d'une fabrique `FabriqueAction` afin de renvoyer effectivement des actions.
3. Définir enfin une classe `Jeu`, qui aura comme attributs un royaume, une fabrique et un joueur. On écrira (et utilisera dans la méthode `main`) les méthodes suivantes :
   — `initialize()` qui initialise le jeu avec un nouveau royaume possédant 0 écus en caisse, un village et un paysan.
   — `jeu(int nombreTours)` qui effectue `nombreTours` tours d'interaction avec le joueur
   — `fin()` qui affiche le résultat final.
4. Tester votre jeu, et essayer de trouver une stratégie pour faire prospérer votre royaume. En déduire une façon d'implémenter un joueur `JoueurAuto` qui joue automatiquement.

Exercice 4 (Bonus) *Variantes & extensions*

Si tout ce que vous avez implémenté jusque là fonctionne, vous pouvez commencer réfléchir à différentes extensions pour le rendre plus complexe et plus intéressant. Voici quelques pistes, mais n'hésitez pas à imaginer les vôtres !

Les villages Jusqu'ici, les villages n'ont pas réellement d'utilité. Vous pouvez imaginer plusieurs manières de les rendre plus intéressants, notamment en faisant en sorte que la production soit plus rentable avec plusieurs villages :

— en modifiant la fonction de production, afin qu'un paysan produise moins lorsqu'il est dans un village trop peuplé ;
— en rajoutant des personnages, par exemple un clerc qui à chaque tour bénit la récolte d'un paysan au hasard parmi ceux du village ;
— etc...

Les personnages Nous n'avons utilisé jusqu'à présent qu'un petit fragment de ce que nous avions défini dans les TPs précédents. Il vous est donc assez aisé de rajouter d'autres personnages à un royaume, par exemple :

— le clergé et différents type de Dieux, et faire prélever la dîme par le clergé ;
— des nobles et entre autres des chevaliers, pour constituer une armée ;
— différents types de roturiers, aux coûts et capacités de productions différents.

Vous penserez bien à ajouter les méthodes nécessaires à votre fabrique ainsi que les classes d'action nécessaires.

Le royaume Enfin, vous pouvez aussi ajouter des événements affectant tout le royaume, et nécessitant la présence de certains personnages pour en atténuer les effets :

— vous pouvez simuler des attaques extérieures, contrables grâce à votre armée ;
— vous pouvez ajouter des famines, épidémies, et autres années d'abondance,
— etc...

Toutes vos idées sont les bienvenues, il y a de toute façon de la marge avant d'atteindre la complexité d'Age of Empires !