Architecture logicielle
Feuille 5
Stratégie temps réel – Armées sous contrôle et dans les âges

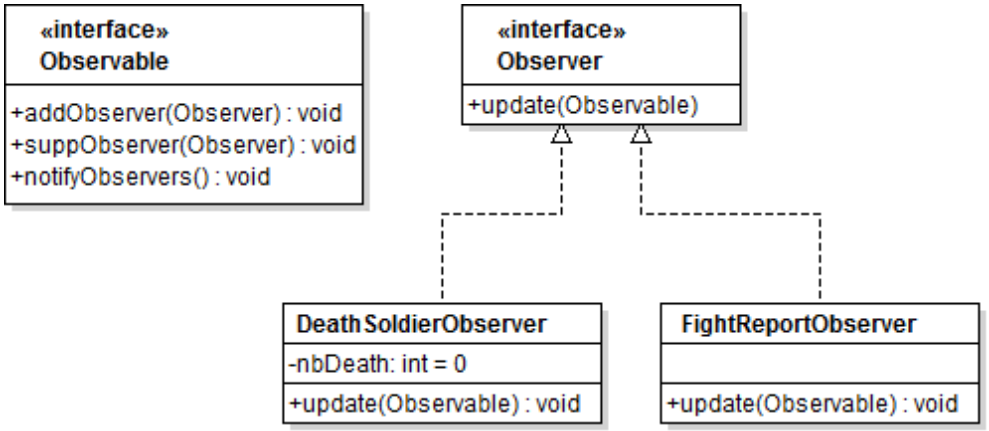
Benoit Barthes - Anne-laure Mesure

December 16, 2014

L'objectif de ce TP est de faire évoluer notre structure actuelle afin de suivre le déroulement d'un combat et de pouvoir gérer des armées d'époques différentes.

Exercice 1

Dans une premier temps, l'objectif est de pouvoir compter le nombre de morts lors d'une bataille et d'afficher le nom de ces soldats tombés ainsi que le nom des groupes armées décimées. On souhaite donc pouvoir observer l'état d'un soldat/groupe armé pour que dès que ses points de vies tombent à 0, on mette à jour un compteur des morts et on affiche la mort du de l'élément.
Pour cela on va mettre en place un pattern observer. On va mettre en place deux observateurs: un compteur de soldats tombés et un afficheur des soldats morts et de groupes armés détruits.
On a donc créé nos propres interfaces Observer et Observable ainsi que les deux observateurs nécessaires.



On a ensuite rendu les classes AbstractSoldierFacade et Army observable. Le point délicat s'est posé lorsqu'il a fallu pouvoir attacher des observateurs à chaque membre d'une armée. Les tests associés à cette implémentation se trouve dans le fichier JUnit TestObserver et execute un combat entre deux soldats et une bataille entre deux armées.

Exercice 2-3

A présent, nous souhaitons pouvoir gérer des armées provenant d'époques différentes, sachant que chaque soldat ne pourra faire que d'un groupe uniquement composés de soldats de son époque et être équipé d'armes de son époque. On va mettre en place deux époques: le moyen-âge et le futur. On aura deux catégories de soldats, les soldats à pied (Infantryman et Furien) et les soldats montés (Horseman et Knightrider), ainsi que trois catégories d'armes, les armes purement offensives (Dagger et Krull), les armes permettant attaques et défenses (Sword et LightSaber) et les boucliers (Shield et HoltzmanShield).
Pour gérer la création de ces familles d'objets on va s'appuyer sur le pattern Abstract Factoy qui permet justement de créer des familles d'objets sans spécifier leurs classes concrètes.

Dans notre implémentation, nous avons créé une interface qui nous permet la création de soldats et d'armes. Deux classes implémentent cette interface, une factory pour le moyen-âge (AbstractFactoryRtsOfPast) et une factory pour le future (AbstractFactoryRtsOfFuture). Pour éviter les anachronismes on a utilisé le pattern Singleton et créé la classe FactoryOfAbstractFactoryRts qui ne qui ne peut avoir qu'une seule instance et empêche d'avoir plusieurs instances de factory d'époques différentes en même temps.