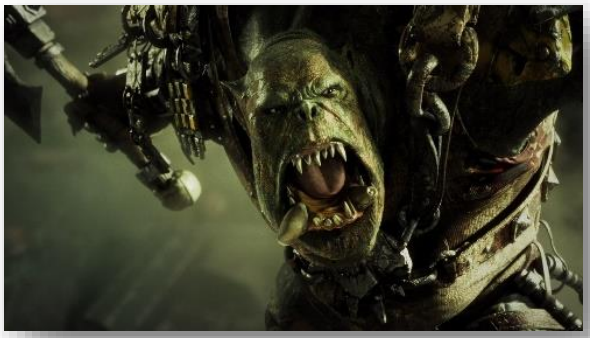
Patron #9 : Prototype

Mise en contexte



Cet exemple de jeu vidéo met en contexte une classe avec un temps de chargement très long (Monstre) et avec un nombre d'instances énormes (les tributs).

Le « main » permet de charger deux tributs

- Une première tribut de 8 orkoides de façon « traditionnelle » (*des new*)
- Une seconde tribut de 6000 orkoides mais avec le patron « prototype »

L'utilisation de la méthode « clone » en Java n'est pas exactement l'équivalent du constructeur par copie du C++ considérant qu'un objet est toujours passé en référence. « Clone » n'est pratiquement jamais appelée en Java en raison des risques liés à la copie en profondeur (deepCopy).

Le code « prototype » fourni est complet et fonctionnel. Il permet de voir une utilisation adéquate du clonage « protégé ». Considérant que vous ne pourriez normalement pas créer vos classes dans le package « spawn », ce code devrait permettre de constater les facettes de code suivante :

- Il est impossible de cloner un monstre sans passer par le « Spawner »
- Le « Spawner » ne peut être hérité que par des classes du même package et la méthode « spawnMonstre » est final protected
- La méthode « clone » est protected final dans les classes enfants (ex : ork »)
- L'interface « Cloneable » indique que les objets sont « propres » au clonage. Ce n'est pas obligatoire et n'affecte pas réellement le programme mais c'est une bonne pratique qui lance le signal clair « *HEY, un bon programmeur certifie que je peux être clonée* »

Prenez le temps de lire et de comprendre la structure du code, pour le plaisir seulement ;-)

Contraintes

À partir du projet « prototype fourni », ajouter les facettes suivantes

- Un nouveau SpawnerXXXXXX de votre choix
- Trois monstres qui pourront être crées par votre « SpawnerXXXXXX » (créer un package spawn.XXXXXX)
- Ajouter les méthodes « genererXXXXXX () » et « genere XXXXXXClone() » dans le main

Compiler et savourer la différence de l'instanciation via une source de données ou via le clonage (copie)