TP n°7

Typage dynamique et polymorphisme

1 Le `.bashrc`

Le fichier `.bashrc` (ne pas oublier le point, signifiant que c'est un fichier caché sous Linux) est un script exécuté à chaque fois que votre terminal est ouvert. Pour automatiser certaines tâches, il est possible d'y ajouter des commandes à exécuter. Dans notre cas, nous allons spécifier le répertoire contenant nos packages à l'aide de la commande *export* utilisée depuis plusieurs TP. Ouvrez un éditeur (*emacs* ou *gedit*) et créez le fichier (ou ouvrez-le s'il existe déjà) : il doit être créé à la racine de votre compte (raccourcis "∼"). Dans ce fichier, ajoutez la commande suivante :

```
export CLASSPATH=.:~/Info0201/packages/
```

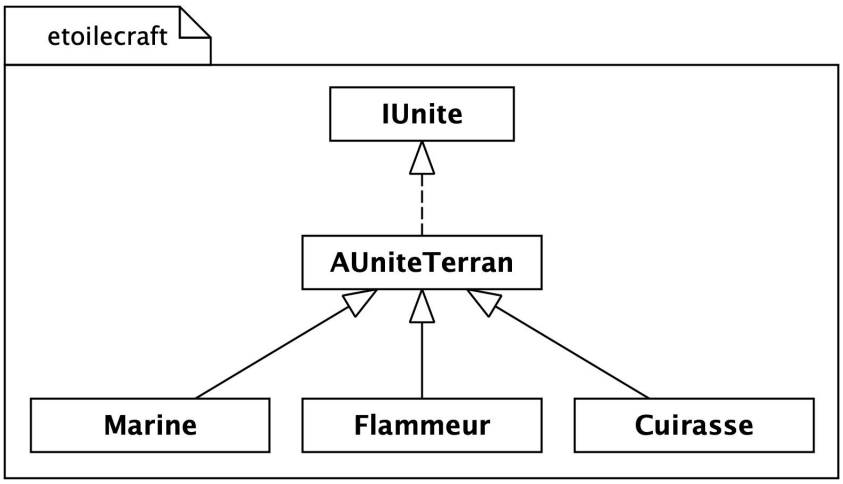
Assurez-vous que le chemin que vous entrez est correct. En particulier, faites attention aux majuscules et prohibez tout espace dans les noms des répertoires (et des fichiers).

Après avoir sauvegardé le fichier, ouvrez un nouveau terminal. Vous pouvez vérifier la présence du fichier `.bashrc` à l'aide de la commande `ls -a` (l'option `-a` permet d'afficher les fichiers cachés). Pour vous assurer que la commande a réussi, tapez la commande `echo $CLASSPATH`. Le contenu de la variable d'environnement `$CLASSPATH` (créée à l'aide de la commande *export*) doit s'afficher, devant correspondre au chemin spécifié dans le fichier `.bashrc`.
À partir de maintenant, vous pouvez compiler les classes de test sans avoir à spécifier le chemin des packages (en n'oubliant pas pour autant le import !!!).

Si des modifications sont apportées au fichier .bashrc , elles ne seront répercutées qu'au prochain démarrage du terminal. Pour éviter de fermer le terminal et d'en ouvrir un nouveau, vous pouvez aussi exécuter le fichier en tapant la commande suivante : `source /.bashrc`

2 Etoilecraft : les Terrans

Nous souhaitons modéliser un jeu de stratégie bien connu appelé *Etoilecraft* dans lequel évoluent trois races différentes, dont les *Terrans*. Dans ce jeu, il existe un ensemble d'unités caractérisées par leurs points de vie et leur point d'armure. Les unités peuvent attaquer d'autres unités. Cependant, certaines unités évoluent dans les airs et les autres sur le sol. Or, une unité peut attaquer soit uniquement au sol, soit uniquement dans les airs, voire les deux. Une unité est donc caractérisée par une puissance de feu sol et une puissance de feu air. Nous considérons le package `etoilecraft` (à télécharger sur *Moodle*) suivant :



La classe `AUniteTerran` est la classe mère de toutes les unités de la race `Terran` :

- Une unité est caractérisée par ses points de vie, ses points d'armure, sa puissance de tir sol et sa puissance de tir air (des réels).
- La méthode `estVolante` est abstraite.
- Une méthode `subirAttaque` inflige des dégâts à l'unité. Ces dégâts sont limités par les points d'armure (une simple division). Ainsi, si une unité reçoit 5.5 points de dégât et qu'elle possède 2 points d'armure, elle ne perd que 2.75 points de vie. Cette méthode retourne `true` si l'unité ne possède plus de point de vie.
- La méthode `attaquer` permet d'attaquer une unité donnée passée en paramètre. Les dégâts sont calculés aléatoirement en fonction de la puissance de tir (sol ou air, suivant la cible visée). Ainsi, si une unité possède une puissance de tir de 5, les dégâts infligés sont compris dans l'intervalle [0 ;5[. La méthode retourne `true` si l'unité est tuée.

Un `Marine` est une unité au sol qui attaque aussi bien au sol, qu'en l'air. Ses puissances de feu sont identiques. Un `Flammeur` est une unité au sol qui n'attaque qu'au sol. Un `Cuirasse` (cuirassé) est une unité volante qui attaque au sol et en l'air. Sa puissance de feu air est inférieure à sa puissance de feu sol.

1. Sur une feuille de papier (ou avec un logiciel de dessin), réalisez le diagramme de classes détaillé. Prenez une photo de votre diagramme que vous rendrez avec l'ensemble des fichiers produits.
2. Écrivez les différentes classes en présence.
3. Écrivez une classe de test `TestEtoileCraft` :
   - Déclarez deux variables de type `IUnite` et affectez-leur des références vers des objets quelconques.
   - Tant que l'une des deux unités n'est pas tuée, la première attaque la deuxième, puis l'inverse.
   - Vous afficherez les unités au fur-et-à-mesure de l'attaque.
   - Pour tester avec d'autres unités, que doit-on modifier ?

3 Typage dynamique et transtypage

Soient les instructions suivantes (en remplaçant les '...' par les bons paramètres), utilisez-les dans la classe de test et corrigez-les au besoin :

1. `AUniteTerran ref1 = new Marine(...); IUnite ref2 = ref1;` Est-ce correct ? Si non, comment est-il possible de corriger la deuxième instruction pour que ça fonctionne ?
2. `AUniteTerran ref1 = new Marine(...); AUniteTerran ref2 = ref1;` Est-ce correct ? Si non, comment est-il possible de corriger la deuxième instruction pour que ça fonctionne ?
3. `AUniteTerran ref1 = new Marine(...); Marine ref2 = ref1;` Est-ce correct ? Si non, comment est-il possible de corriger la deuxième instruction pour que ça fonctionne ?