**Travail pratique 1 – Duel Liste de contrôle**

Vous devez compléter et remettre cette liste de contrôle avec votre travail.

*Lorsque vous ne pouvez pas cocher une case, vous devez l’expliquer avec un commentaire.*

**Exigences de fonctionnalités**  Il est possible de :

x Créer un guerrier;

x Tout en respectant les contraintes énoncées au niveau de ses aptitudes. Il est possible de :

x Créer un athlète;

x Tout en respectant les contraintes énoncées au niveau de ses aptitudes. Il est possible de :

x Créer un magicien;

x Tout en respectant les contraintes énoncées au niveau de ses aptitudes.

x Il est possible de créer des capacités en spécifiant leurs caractéristiques.

x Un combattant peut posséder n’importe quelle capacité, même si elles ne sont pas les plus adaptées pour lui.

x Il est possible d’évaluer la puissance d’une capacité en fonction de ses caractéristiques et des aptitudes du combattant qui la met en œuvre.

x Un combattant peut provoquer en duel un autre combattant, s’il utilise une capacité d’attaque.

x Un combattant peut capituler et refuser de combattre en duel.

x Un combattant provoqué en duel peut riposter avec une capacité d’attaque.

x Un combattant provoqué en duel peut riposter avec une capacité de défense.

x Lorsqu’il y a combat dans un duel, le perdant perd des points de vie, conformément au calcul énoncé.

À la fin du duel, conformément aux règles énoncées :

x Le gagnant reçoit ses récompenses : augmentation de ses aptitudes et capacité supplémentaire.

X Le perdant est pénalisé : diminution de ses aptitudes.

Un combattant peut aller se soigner à l’infirmerie, s’il a une capacité de soin.

Manque de temps pour implémenter l’infirmerie.

x Un simulateur permet de démontrer clairement la création de capacités, de combattants et le déroulement de duels.

**Exigences de qualité**

# Code

x Nommage : toutes les méthodes, les classes, les attributs, les variables et les constantes ont un nom significatif indiquant leur raison d’être ou leur contenu selon le cas.

Il n’y a pas de conditions complexes (elles sont encapsulées).

x Il n’y aucun chiffre magique.

x Il n’y a aucune chaîne de caractères magique.

x Les commentaires au travers du code sont réellement informatifs.

x Les conventions Java et les normes de programmation, telles que montrées, sont respectées.

x Toutes les classes sont petites (moins de 100 lignes).

x Toutes les méthodes sont les plus petites possibles (moins de 15 lignes).

x Les méthodes ont peu de paramètres.

x L’indentation du code est irréprochable.

x Dans l’ensemble, le code est clair, facile à lire et à comprendre.

X Dans l’ensemble, le code est simple (KISS : Keep it Simple, Stupid) Il n’y aucune erreur ni aucun avertissement lors de la compilation.

Note : si vous jugez qu’un avertissement n’a pas lieu d’être, consultez votre professeur et faites taire l’avertissement manuellement.

x Le simulateur s’exécute sans erreur.

# Exceptions

x Des exceptions sont utilisées à la place de valeurs de retour codées ou de valeur de retour null.

x Des exceptions sont utilisées quand le flot normal du code est interrompu.

x Des exceptions « Runtime » sont utilisées pour gérer les erreurs de programmation. Les exceptions sont les plus spécifiques possibles.

# Tests

x Toutes les méthodes de tests ont un nom significatif indiquant ce qui est testé, quitte à avoir un nom de méthode long.

x Chaque méthode de test ne teste qu’un seul comportement.

x Chaque méthode de test respecte l’organisation : Arrange/Act/Assert

x Des objets simulés (« mocks ») ont été utilisés dans les tests au besoin.

Combien de tests sont exécutés ? 42

# Conception objet

x Encapsulation : toutes les propriétés d’une classe (exceptées les constantes) sont privées xLe principe de conception suivant est respecté: Single Responsability Principle.

x Le principe de conception suivant est respecté: Loi de Demeter.

x Le principe de conception suivant est respecté: Query-Command Separation.

x Le principe de conception suivant est respecté: Tell, don’t ask.

x Héritage : les super-classes ne connaissent pas leurs enfants.

Polymorphisme : le mot clé « instanceof » n’a pas été utilisé.

Je n’ai pas réussi à implémenter le polymorphisme.

Polymorphisme : lorsque possible, le polymorphisme a été préféré à If/Else ou Switch/Case

(éviter les if).

Je n’ai pas réussi à implémenter le polymorphisme.

Le principe de conception suivant est respecté: programmer avec des interfaces (ou des classes abstraites) pas avec leurs implémentations.

Pas respecter partout, comme pour la classe Fighter.

Le patron de conception suivant est utilisé : Strategy pattern.

Précisez: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

La pratique de conception suivante est utilisée : Simple Factory.

Précisez: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

x Si une nouvelle race de combattant est proposée dans le jeu, il est possible de développer cette nouvelle fonctionnalité dans le code sans modifier le code des classes existantes.

Si non : quelles classes seraient à modifier : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

X Si une nouvelle capacité est proposée dans le jeu, il est possible de développer cette nouvelle fonctionnalité dans le code sans modifier le code des classes existantes.

Si non : quelles classes seraient à modifier : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_