TD n°16 – Héritage, polymorphisme, collections
Le projet Pokemons

Rappel des consignes

- **Ce projet se réalise sur plusieurs séances, par groupe de 2 étudiants.**
- **Votre enseignant responsable en TD vous donnera ses indications de rendu.**
- **Le code doit être correctement commenté (pas trop, ni trop peu), robuste (gestion des exceptions), et associé à des TestCase JUnit.**
- **Un travail minimum est demandé lors de chaque séance de TD. La réalisation de ce travail est obligatoire et sa bonne réalisation assure la moyenne au groupe.**
- **Des extensions seront suggérées ou laissées au libre choix des étudiants. Toute extension correctement implémentée améliore la note du groupe.**
- **Au contraire, tout code similaire entre groupes diminue la note de tous les groupes concernés.**
- **Lors du DS, une partie des questions portera sur ce projet.**

➢ Pas de nouvelle partie obligatoire

Assurez-vous d'avoir terminé les parties obligatoires des TDs précédents.

Une fois que tout fonctionne, vérifier que votre code est clair, commenté, robuste, et associé à des TestCase JUnit.

Une fois toutes ces vérifications faites, vous pouvez passer aux extensions. Vous êtes libres de développer ces extensions en ajoutant les éléments que vous jugez nécessaires (attributs, méthodes), **mais sans supprimer les fonctionnalités de base**.

➢ Extensions possibles (Javadoc + Rappel des extensions proposées précédemment)

Générer une javadoc

Javadoc est un outil fourni avec le JDK pour permettre la génération d'une documentation technique à partir du code source.

Le but de cette extension est de créer une documentation technique de votre projet.

Dans le code source Java, vous devez définir des commentaires utilisant des balises *@tag* reconnues par javadoc. Un exemple de commentaires javadoc est disponible dans la classe `Point.java` du TD3 dans SupportCours.

Afin de générer la doc dans Eclipse, suivre le tutoriel suivant : http://www.objis.com/formation-java/tutoriel-java-creation-javadoc-eclipse.html

- Commenter toutes les classes de votre projet en utilisant les balises reconnues,
- Générer la documentation en ligne dans votre Java Project Eclipse.

Lien vers la page officielle Oracle décrivant comment générer la documentation à l'aide de la commande javadoc : http://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

Evolutions des Pokemons

Un joueur peut maintenant faire évoluer un Pokemon. Cela correspond à enlever le Pokemon que l'on souhaite évoluer de la collection et ajouter son « évolution » (qui est un autre Pokemon de même type, en général plus gros et plus puissant).

Pour cette dernière fonction, il est nécessaire de connaître les évolutions possibles des Pokemons. Un fichier texte nommé *EvolutionPokemon.txt* est disponible en annexe de ce sujet. Il contient sur chaque ligne le nom du Pokemon, suivi du nom de son évolution.

Afin de modéliser l'évolution des Pokemons, on enrichit la classe `Partie` d'une table d'associations permettant d'associer pour chaque nom de Pokemon, le nom de son évolution. Elle est initialisée dans une méthode qui lit le contenu du fichier et remplit la table d'association.

Ensuite, on considère qu'un Pokemon qui a évolué :

- Prend le nom de son évolution ;
- Garde le même type ;
- Voit ses autres caractéristiques évoluer de la manière suivante :
    - La taille augmente de 20% ;
    - Le poids augmente de 15% ;
    - Les points de vie ne changent pas ;
    - La puissance d'attaque augmente de 10%.

Les autres caractéristiques ne changent pas.

Enrichir la classe `Joueur` afin de prendre en compte ces nouvelles fonctionnalités.

Gestion des points et niveaux d'un joueur

Les attributs niveau et nombre de points des joueurs ne sont pour l'instant pas utilisés.

Enrichir la classe `Joueur` en ajoutant une méthode `gainNiveau()` qui augmente le niveau d'un joueur à chaque fois qu'il atteint 100 points (le nombre de points est alors réinitialisé).

Chaque joueur engrange des points en effectuant des actions avec les Pokemons :

- Attraper un Pokemon fait gagner 3 points ;
- Transférer un Pokemon fait gagner 1 point ;
- Evoluer un Pokemon fait gagner 10 points.
- Attaquer un Pokemon rapporte 5 points au joueur s'il inflige + de dégâts à son adversaire que ce qui lui est infligé.
  Le joueur qui se défend n'engrange pas de points.

Mettre à jour votre code afin d'intégrer cette fonctionnalité. Attention à vérifier à chaque fois si le gain des points fait passer au niveau supérieur ou pas.

Varier les Pokemons

- Pour construire la collection initiale des joueurs, les données fournies dans le fichier peuvent être modifiées comme suit :
    - La taille, le poids, les points de vie et les points de combat du Pokemon tiré aléatoirement peuvent être modifiés de + ou – 20% pour générer plus de Pokemons différents.

Enrichir les combats

Le choix du Pokemon dans la méthode attaquer peut être enrichi de la manière suivante :

- Une méthode `randomAttaque` permet comme avant de choisir au hasard un Pokemon dans sa collection pour attaquer (de manière équivalente `randomDefense` choisit au hasard le Pokemon du joueur qui doit se défendre dans sa collection).
- Une deuxième méthode `bestAttaque` (respectivment `bestDefense`) choisit optimalement le Pokemon d'attaque (resp. de défense). Les critères sont :
    - Pour l'attaque, il peut être judicieux de choisir son meilleur Pokemon pour attaquer (i.e. celui qui a la meilleure attaque, ou celui qui a le plus de points de vie,…).
    - Pour la défense, il peut être judicieux de choisir le meilleur Pokemon pour le combat, en fonction des caractéristiques du Pokemon déjà choisi par le Joueur attaquant.

Enrichir les modes de jeu.

Vous pouvez par exemple définir des parties de la manière suivante :

- Partie « ordonnée » :
    - Au premier tour, le joueur qui attaque est le premier dans l'ordre naturel des joueurs.
    - Puis, le joueur qui a perdu le moins de points de vie au combat précédent est celui qui attaque au tour suivant.
- Partie « plus fort d'abord » où le Pokemon qui attaque à chaque tour est celui qui a la plus grande puissance parmi les Pokemons toujours en vie de la collection du joueur (en les comparant et/ou triant la collection avec l'ordre naturel ou un ordre temporaire).
- Partie « plus vivant d'abord » où le principe est le même que ci-dessus mais le critère de choix est le nombre de points de vie et non la puissance.
- Partie « interactive » où chaque joueur décide à chaque tour quel Pokemon il va utiliser pour attaquer l'adversaire.
- Partie à *n* joueurs (dans tous les modes) : quelle structure de données utiliser pour les joueurs ?