TD n°14 – 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.**

Les joueurs

➢ 2ème partie obligatoire

Collection de Pokemons

Nous allons maintenant considérer la classe `Joueur` qui contient (au moins) :

- Un nom ;
- Un niveau ;
- Un nombre de points ;
- Une collection de Pokemons ;

Quel est le type de collections à utiliser pour déclarer la collection du joueur ? (« Généraliser » le plus possible le type déclaré)

- La méthode `vitesseMoyenne()` retournant la vitesse moyenne des Pokemons de la collection ;
- Des méthodes `vitesseMoyenneTYPE(String t)` calculant la vitesse moyenne des Pokemons de type `t` de la collection.
  Lever des exceptions !
- La méthode `toString()` affichant l'état d'un joueur.

En plus de ces fonctionnalités de base, un joueur peut :

- Attraper un Pokemon en l'ajoutant à sa collection ;
- Transférer un Pokemon en l'enlevant de sa collection ;

Combat de Pokemons entre Joueurs : méthode aléatoire

Enfin, un joueur peut attaquer avec l'un des Pokemons de sa collection un autre joueur. Le joueur attaqué choisit alors un Pokemon de sa collection qui combattra contre le Pokemon de l'attaquant.

Nous souhaitons définir une méthode `void attaquer(Joueur adversaire)`.

1. **Etant donnée l'existence d'une méthode `attaquer` dans les classes des Pokemons, créer une interface `IAttaque` contenant la déclaration factorisée de la méthode `attaquer`.**
2. **Mettre à jour les classes des Pokemons qui doivent maintenant implémenter l'interface `IAttaque`.**
3. **Ajouter l'implémentation de l'interface dans la classe `Joueur`.**

L'attaque la plus simple est de choisir aléatoirement les Pokemons qui se combattent dans les collections respectives des 2 joueurs.

La méthode `attaquer` de la classe `Joueur` doit donc :

- Choisir aléatoirement un Pokemon dans la collection de chacun des joueurs ;
- Appeler la méthode `attaquer` de la classe `Pokemon` afin de modifier les points de vie des Pokemons qui combattent.

Les points de vie des 2 Pokemons en combat sont affectés (on considère la défense comme une attaque). Vous pouvez au choix :

- *Infliger les dégâts aux 2 Pokemons en combat dans la méthode attaquer ;*
- *Créer des fonctions de défense pour infliger les dégats du Pokemon qui attaque initialement.*

Comparer des Pokemons et des joueurs

On souhaite pouvoir comparer des joueurs entre eux en fonction de leur niveau ou des Pokemons qu'ils possèdent dans leur collection (de même il faut pouvoir classer les Pokemons entre eux et donc les comparer).

Le but est de déterminer a priori si des joueurs sont pressentis pour gagner afin de, pourquoi pas, calculer des côtes et lancer des paris sur les combats !

Un ordre naturel sur les Joueurs est défini par l'ordre alphabétique de leur nom.

Un ordre naturel sur les Pokemons est défini par l'ordre alphabétique de leur type. Pour 2 Pokemons de même type, l'ordre dépend de leur point de vie. Un Pokemon se situe avant un autre de même type s'il a plus de points de vie.

- **Définir les ordres précédents en implémentant l'interface `Comparable<T>`.**

Les tests unitaires

Ne pas oublier de tester la classe `Joueur` ! Et agrémenter les tests des Pokemons avec les nouvelles fonctions développées.

➢ Extensions possibles

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` peut choisir 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.

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 `Pokemon` 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 les classes `Joueur` et `Pokemon` 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.