Groupe 2

# Avis général

Le code fonctionne, et il correspond aux spécifications : très bien joué !

Il est correctement découpé, et pour moi c’est le plus important.

Je pense que vous avez accompli tous les objectifs pédagogiques du TP.

Il y a des remarques ci-dessous qui ne remettent pas en cause mon jugement sur votre très bonne maîtrise du Java, et (je pense) du travail en équipe. Je vous mets ces remarques pour que vous vous amélioriez encore.

Merci pour votre documentation : bon travail là aussi.

# Conseils généraux

N’hésitez pas à demander à l’IDE d’organiser les imports pour nettoyer les imports inutiles. Vous pouvez aussi forcer l’auto import à chaque sauvegarde (ce que font beaucoup d’entreprise). Cherchez en anglais : « organize imports » pour activer/désactiver l’auto import à chaque sauvegarde (si l’IDE est en français, la chaîne de rechercher doit être « organiser les imports »).

Merci pour les diagrammes. Pour le diagramme de classes, une relation unidirectionnelle n’a qu’une seule cardinalité (celle de la direction de la relation). Même si le diagramme de classes a été casse pieds à maintenir, en avoir un au départ et « à peu près en état » ensuite vous a aidé (je pense) et peut être donné des idées quant à son utilité.

Pour vos problèmes de Git, vous utiliserez sûrement en entreprise un flux git ou « Git flow ». Celui-ci vous imposera (pour résumer) de créer une branche par User Story ou fonctionnalité. Les conflits seront toujours présents, quelle que soit la méthodologie ou l’outil utilisé.

Super que le debugger vous ait été utile !! Ce n’est pas la dernière fois:)

Vos axes d’amélioration sont bien pensés je trouve.

# Remarques sur les classes

* Arene (et ses classes filles): Utilisation de l’héritage, ce qui est très bien. On pourrait faire en sorte qu’Arene implémente des méthodes effetDebutCombat et effetTourDeCombat qui ne font rien, ce qui simplifierait le code des classes filles.
* Combat : très bien pour les algorithmes, c’est la classe la plus compliquée à gérer normalement à ce niveau là. Elle encapsule bien le comportement des combats. Bravo pour la possibilité que l’arène amène à un match nul, que je n’avais même pas imaginée !
* Simulation : très bien aussi pour les algorithmes gérant l’entrée sortie. Le code est très clair, sûr et efficace. La boucle while utilise un label. Pourquoi pas, mais c’est rarement utilisé en Java (parce que considéré comme étant la porte ouverte à du code ‘spaghetti’).
* InvalidPokemonTypeException : elle hérite de Runtime, c’est très bien. Elle remontera jusqu’au main (et même au dessus) en cas de problème , ce qui arrêtera le programme.
* Classes de test : très bonne idée. En Java, pour info, JUnit est très utilisé pour structurer ce genre de code.
* API : Classe bien pensée, qui fait ce qu’elle a à faire : gérer le ‘stock’ des pokémons de l’entreprise. Pour pinailler, on pourrait transformer pas mal de commentaires en JavaDoc pour faciliter le travail des utilisateurs de la classe, et changer getPokemonByID(int) en getPokemonByID(Integer) et se débarasser d’un cast.
* Attaque : le code calculant les bonus est bien encapsulé. L’utilisation des maps est très bien pour calculer les bonus. Si l’on voulait encore plus encapsuler (et ainsi éviter aux développeurs utilisant Attaque de faire des erreurs de calcul), cette méthode pourrait fournir une méthode qui calcule les dégâts totaux d’une attaque, puisqu’elle contient tous les attributs pour ce faire.
* Classes filles d’attaque : impeccable, tout est bien encapsulé
* Pokemon : contient des attributs final, du static pour le compteur, du protected. Bonne gestion de ces concepts vus dans le cours ! Bref, bloque les accès à bloquer et ouvre les accès à ouvrir. Allez, on pourrait demander à ce que incrementNbPokemonCrees() soit private pour éviter à un ptit malin d’incrémenter le compteur à notre place. La méthode getNbPokemonCrees pourrait être static, vue qu’elle n’utilise que des attributs static. Ceci permettrait à n’importe qui de faire Pokemon.getNbPokemonCrees. Ah, par contre, je supprimerais la méthode setNbPokemonCrees pour éviter que le compteur soit corrompu. (un ‘reset’ du compteur pourrait avoir des effets non désirés sur le référentiel). Le reste des getters et setters, c’est très bien.
* Classes filles de Pokemon : impeccable.
* TypePokemon : bonne utilisation de l’enum.