TP3-4 Pokedeck

Sommaire

[Analyse 2](#_Toc406456949)

[Modelisation – Architecture 3](#_Toc406456950)

[Les fonctionnalités implémentées 4](#_Toc406456951)

[Les choses à améliorer et à faire 4](#_Toc406456952)

[Bilan 5](#_Toc406456953)

# Analyse

Le Pokedeck est une application devant contenir une liste de cartes pokemons et leurs caractéristiques. Cela ressemble à un inventaire ayant la possibilité d’enregistrer toutes les cartes existantes et de faire un paquet de cartes (deck) à jouer.

D’après la documentation fournie, les cartes sont classées par type : Pokemon, Trainer et Energy.

Les pokemons peuvent comporter des évolutions. Les évolutions sont considérées aussi comme des pokemons. Or il existe des cartes qui illustrent des méga-évolutions" « ex mega ».

Il y a dans le jeu, des cartes spéciales « ex ». Les méga-évolutions sont justement des cartes ex.

Les cartes « Trainers » sont des cartes apportant du soutien ou un bonus au combat. Il existe plusieurs sortes (items, supporters et stadiums).

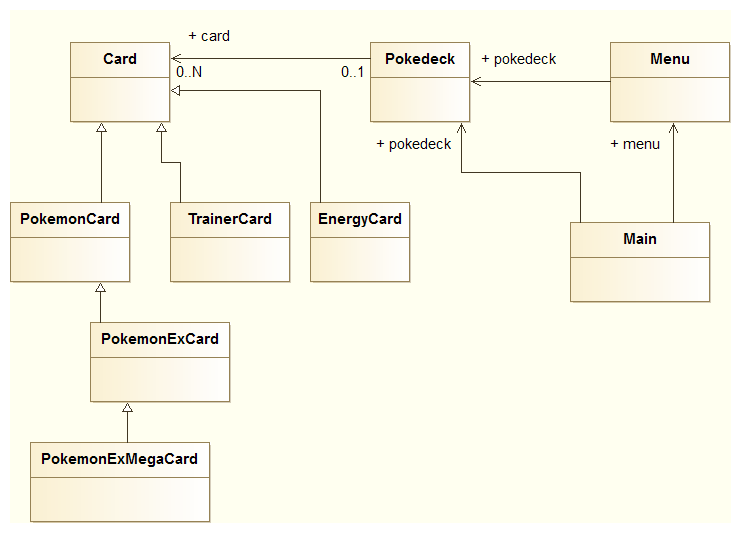
Puis il y a les cartes « energy » qui doivent être utilisées dans le jeu pour les pokemons associés au même élément. Ces monstres ont en effet tous un type d’élément (fire, water, grass etc…).

Mon application prend en compte les aspects énoncés ci-dessus. Après une étude du sujet, on en déduit donc :

Une classe père/racine est la carte à jouer. Les Pokemons, Trainers et Energy héritent de la classe carte. Les cartes spéciales « ex » héritet de Pokemon et « ex mega » de « ex ». L’architecture que j’ai choisi d’implémenter sera présentée dans la partie modélisation. En choisissant d’utiliser le polymorphisme, mon code pourra en principe gérer les instances de cartes quelque soit son type. Par exemple quand l’utilisateur décide de créer une carte avec un type spécifié, le programme est adapté pour récupérer cette carte et créer l’instance en fonction de sont types qui n’est d’autres que des objets issues des classes fils de « carte ».

Un classe Pokedeck contiendra une arrayList de cartes. Le fait que toutes les classes héritent de cartes permet facilement d’ajouter et de travailler sur ces éléments. Les classes sont majoritairement structurées de manière simple avec le moins d’accesseur possible. Seule la fonction toString sera implémentée pour pouvoir afficher les données.

# Modelisation – Architecture



# Les fonctionnalités implémentées

L’**IHM** en mode console a été faites dans ce TP.

**Ajout** **d’une nouvelle carte** en précisant les propriétés par l’utilisateur

Possibilité de créer des cartes Pokemon, Ex Pokemon, Pokemon Ex Mega Evolué, Trainer, Energy. Chaque type de cartes ont des sous types ou énergies. Ces propriétés sont demandées à l’utilisateur.

Exemple : Element fire, water, grass etc….

Exemple : Trainer de type Item, supporters ou stadiums

**Ajout d’une description** pour une carte, l’utilisateur saisit le nom de la carte pour pouvoir l’updater en écrivant une description.

**Supprimer** une carte en précisant son nom

**Consultation** de la collection de carte enregistrée dans le pokedeck

**Chercher les cartes par nom**, cela affiche les cartes comportant le nom saisi

**Chercher les carte par type**, cela affiche les cartes qui sont associée au type saisi

**Sauvegarde,** la classe carte et pokedeck sont Serialisable afin de pouvoir enregistrer les données vers un fichier texte. C’est le pokedeck contenant la collection de cartes qui est sauvegardé.

Le pokedeck est initialisé pour que cela soit plus facile à tester/debugger (voir fonction ini()). Le mieux est de ne pas mettre de cartes initialisées dans le pokedeck au préalable car le jeu demande normalement à l’utilisateur de faire sa collection en partant de zéro.

# Les choses à améliorer et à faire

La fonction **Update** a une ébauche de code mais doit être écrite de façon à donner la possibilité de mettre à jour les propriétés d’une carte.

L’IHM est redondant, trouver un moyen de mieux gérer les Scanners, il y a eu des difficultés avec cette classe. Il reste à factoriser les fonctions. Séparer par exemple l’affichage et les procédures. Il faudrait peut-être que j’utilise des tableaux pour les différents choix possibles (types de carte, types d’énergie…).

Faire des fonctions de vérification afin de vérifier les données saisies par l’utilisateur afin d’éviter les possibles erreurs/exceptions.

Les bonus (Maven, POJO) n’ont pas été faits. Chercher comment charger les données d’un pokedeck sauvegardé.

# Bilan

Au niveau de l’architecture, le système de polymorphisme m’a facilité la tâche pour travailler sur les types de cartes. De plus, cela n’a pas été difficile de rajouter des fonctions métiers car la classes Pokedeck pouvait appeler les méthodes de la classe Carte sans se soucier du type de la carte. Je n’ai donc pas changé l’architecture de mon projet. Je pense que l’étude du sujet est primordiale avant de commencer un projet. Faire un diagramme UML permet de visualiser la structure du code puis de savoir par quelle partie commencer. Les fonctionnalités à implémenter paraissent plus claires.