Gabriel DEGEZ

Programmation Orientée Objet en Java

Modélisation et Implémentation d’un Pokédeck

L3 Professionnelle Projet Web Développement et Communication Multimédia

Année 2018-2019

PLAN

INTRODUCTION :

Il s'agit de décomposer le problème de base en sous-problèmes plus simples à résoudre.

Les interactions entre ces sous-problèmes définissent l'algorithme global de la solution.

L'identification des informations échangées lors de ces interactions permet de définir les objets manipulés (données et structures de données) ainsi que leur portée.

I/ Fonctionnement attendu du logiciel Pokédeck

Le Pokédeck est la collection de cartes (60 maximum) dont dispose un joueur de Pokémon en début de partie. C’est sa « pioche ». Il s’agit de la gérer c’est à dire la constituer avant la partie puis l’utiliser en cours de partie.

1. avant la partie : une phase initialisation
   - soit le joueur sélectionne, une par une, les cartes parmi les milliers qui existent et les ajoute une par une à son deck
   - soit il « charge » dans son deck un ensemble de carte sauvegardées dans un fichier (alimente son deck à partir d’un fichier de sauvegarde)
2. à tout moment de cette phase il doit pouvoir
   - visualiser sa collection (toute la collection)
   - sélectionner un sous ensemble de carte sur plusieurs critère et les visualiser
   - mettre à jour une carte
   - lorsque la collection est finalisée et convient au joueur, il doit mélanger sa collection pour pouvoir débuter la partie.

- En fonction du déroulement du jeu, il doit pouvoir piocher une carte (la retirer de son deck)

3. Enfin il doit pouvoir à tout moment sauvegarder sa collection dans un fichier

II/ Algorithme principal

L'algorithme principal est le suivant :

- création du deck
- boucle infinie avec :
  - affichage du menu principal (cf. infra)
  - exécution de la fonctionnalité du menu choisie
  - on sort de la boucle et du programme si on a décidé de sortir

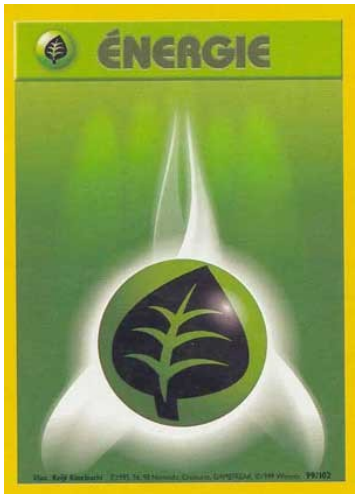
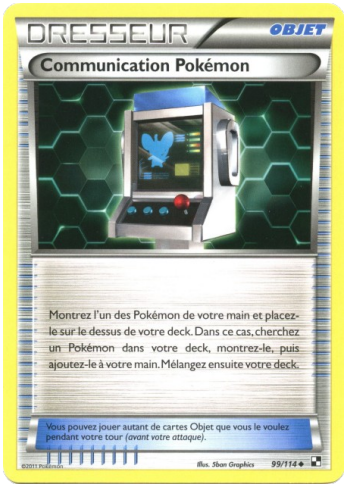
Il repose sur un menu d'accueil interactif permettant de choisir l'opération souhaitée

1. restaurer les cartes à partir d'un fichier
2. ajouter une carte,
3. rechercher une carte ou plusieurs cartes à partir de plusieurs critères
4. afficher une carte
5. modifier une carte
6. afficher la totalité des cartes
7. mélanger les cartes
8. retirer une carte
9. sauvegarder les cartes dans un fichier
10. sortir

III/ Description des classes (attributs et méthodes)

Le Deck : un conteneur de « cartes » :

Ces cartes sont de trois types : « Énergie » / « Dresseur » / « Pokémon ». Suivants ces types, elles peuvent contenir des blocs de données très différentes :

Carte « Énergie »	Carte « Dresseur »	Carte « Pokémon »
		
• Type de carte : energie • type d'énergie (10 + 1 : cf ci dessous)	• Type de carte : dresseur • type de dresseur (3 : objet/stade/supporter)	• Type de carte : pokemon • type de pokemon (énergie)
plante eau feu métal électrique psy combat obscurité fée dragon incolore	• extension *(au gré des évolutions du jeu, des nouvelles cartes ont été ajoutées, ces collections sont groupées en extension qui ont un nom et un numéro)* • numéro carte (alphanum : *en général, c'est le numéro de la carte/nombre de cartes dans l'extension*) • nom carte • règle (dresseur : *les dresseurs de même type suivent la même règle*) • texte (propre à la carte)	• extension • numéro carte (alphanum) • nom carte • description • évolution (base / niveau 1 / niveau 2) • nom évolution • point vie (PV) • type faiblesse • valeur faiblesse (x2/+10/+20/+30/+40) • type résistance • valeur résistance (-20/-30) • point retraite
NB completer le code !		• nom talent • description talent
		• nom attaque • description attaque • type énergie d'attaque • point type attaque • point attaque incolore • coût attaque

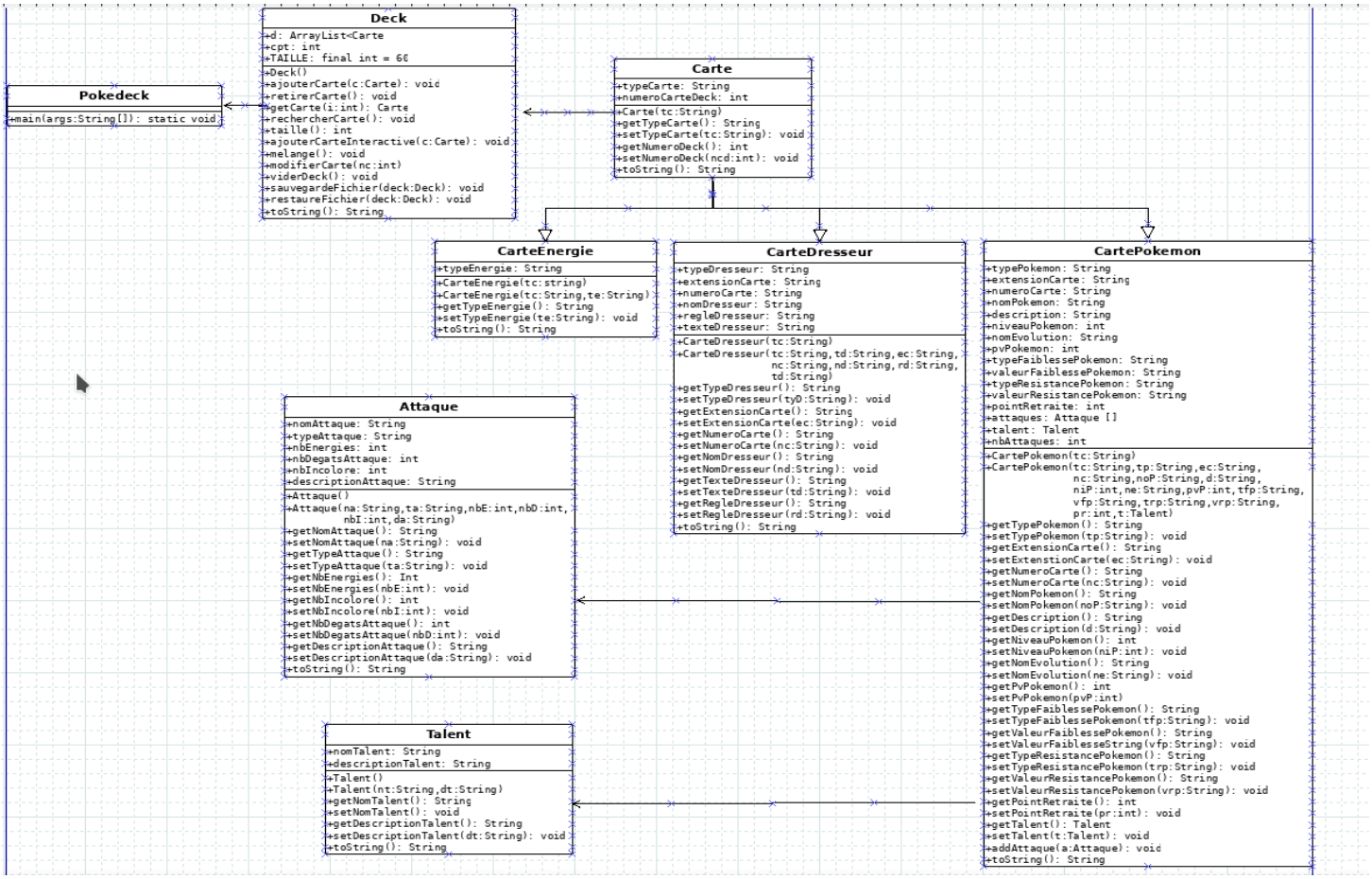
Le tableau met en évidence

- un bloc de données communes, ce qui permet de décrire un objet de type « carte »
- un bloc de données complémentaires spécifiques aux différents types de carte, ce qui permet de décrire des objets de type « carte Énergie », « carte Dresseur » et « carte Pokémon ».
- deux blocs de données complémentaires aux cartes Pokémon pour les « attaques » ou « talents » dont elles peuvent disposer et qui permettent de décrire des objets de types « attaque » ou « talent »

Les différents types de cartes ayant des données différentes, cela suggère l'utilisation d'une classe spécifique pour chaque type de carte. Ces classes héritant elles-même d'une classe « Carte » générique.

De même, on remarque que les « cartes Pokémon » intègrent d'autres sous-ensembles (« Talents » et « Attaques ») pouvant être en nombre variable. Pour faciliter la gestion de cet aspect modulaire, on utilisera aussi une sous-classe distincte pour décrire les entités « Attaque » et « Talent » (cf diagramme des classes plus loin)

Ci-dessous, le diagramme des classes manipulées



1. Carte

Cette classe sert à gérer les données d'une carte communes aux trois types de carte.

Elle n'est définie que par deux attributs :

- `typeCarte` de type **`String`** servant à gérer le type de Carte (Énergie, Dresseur, Pokémon)
- `numeroCarteDeck` de type **`int`** servant à identifier une Carte dans le deck. Cet attribut va faciliter les opérations de recherche et de modification d'une Carte

Méthodes	Entrée	Sortie	Description
`Carte (tc)`	`String`		Constructeur

`getTypeCarte()`
`getNumeroCarteDeck()`
 	
`setNumeroCarteDeck(ncd)`
`setTypeCarte(tc)`
	
	`int`
	`String`
		`String`
		`int`
			Accesseurs
			
			Modificateurs

`toString()`		`String`	Entrée sortie : affichage console de la description de la carte

1. Classe CarteEnergie

Cette classe sert à gérer les données d'une carte Energie et elle hérite de la classe `Carte`.

Elle est définie par un seul attribut `typeEnergie` de type **`String`** servant à gérer le type de Dresseur

Méthodes	Entrée	Sortie	Description

`CarteEnergie (tc)`
 	
`CarteEnergie (tc,te)`
	`String`		Constructeur
			
			Constructeur testé dans une classe de test

`getTypeEnergie()`
 	
`setTypeEnergie(te)`
	
	`String`
			Accesseurs
			
			Modificateurs

`toString()`		`String`	Entrée sortie : affichage console de la description de la carte Énergie

2. CarteDresseur

Cette classe sert à gérer les données d'une carte Dresseur et elle hérite de la classe `Carte`.

Elle est définie par plusieurs attributs :

- `typeDresseur` de type **`String`** servant à gérer le type de Dresseur
- `extensionCarte` de type **`String`** servant à gérer l'extension de la Carte
- `numeroCarte` de type **`String`** servant à gérer le numéro de la Carte Pokémon
- `nomDresseur` de type **`String`** servant à gérer le nom du Pokémon
- `regleDresseur` de type **`String`** servant à gérer la description du Pokémon
- `texteDresseur` de type **`String`** servant à gérer le niveau du Pokémon

Méthodes	Entrée	Sortie	Description

`CarteDresseur(tc)`
 	
`CarteDresseur(tc,tyD,ec,nc,nd,td,rd)`
	`String`	`void`	Constructeur
			
			Constructeur testé dans une classe de test

`getTypeDresseur()`
`getExtensionCarte()`
`getNumeroCarte()`
`getNomDresseur()`
`getTexteDresseur()`
`getRegleDresseur()`
 	
`setTypeDresseur(tyD)`
`setExtensionCarte(ec)`
`setNumeroCarte(nc)`
`setNomDresseur(nd)`
`setTexteDresseur(td)`
`setRegleDresseur(rd)`
	
	`String`
	`String`
	`String`
	`String`
	`String`
	`String`
		`String`
		`String`
		`String`
		`String`
		`String`
		`String`
			Accesseurs
			
			Modificateurs

`toString()`		String	Entrée sortie : affichage console de la description de la carte Dresseur

3. Classe CartePokemon

Cette classe sert à gérer les données d'une carte Pokémon et elle hérite de la classe `Carte`.

Elle est définie par plusieurs attributs :

- `typePokemon` de type **`String`** servant à gérer le type de Pokémon
- `extensionCarte` de type **`String`** servant à gérer l'extension de la Carte
- `numeroCarte` de type **`String`** servant à gérer le numéro de la Carte Pokémon
- `nomPokemon` de type **`String`** servant à gérer le nom du Pokémon
- `description` de type **`String`** servant à gérer la description du Pokémon
- `niveauPokemon` de type **`int`** servant à gérer le niveau du Pokémon
- `pvPokemon` de type **`int`** servant à gérer le point de vie du Pokémon
- `typeFaiblessePokemon` de type **`String`** servant à gérer la description du Pokémon
- `valeurFaiblessePokemon` de type **`String`** servant à gérer la description du Pokémon
- `typeResistancePokemon` de type **`String`** servant à gérer la description du Pokémon
- `valeurResistancePokemon` de type **`String`** servant à gérer la description du Pokémon
- `nomEvolution` de type **`String`** servant à gérer le nom de l'évolution du Pokémon
- `pointRetraite` de type **`int`** servant à gérer le point de retraite du Pokémon
- `attaques` de type **`Attaque[]`**(tableau d'**`Attaque`** dont la classe est définie plus loin) servant à stocker les attaques du Pokémon (jusqu'à trois maximum). Ce tableau est initialisé dans les deux constructeurs.
- `talent` de type **`Talent`** (classe définie plus loin) servant à gérer le talent du Pokémon (un seul maximum)
- `nbAttaques` de type **`int`** servant à gérer le nombre d'attaques du Pokémon stockées dans le tableau attaques et incrémenté à chaque ajout d'une attaque dans ce tableau.

Méthodes	Entrée	Sortie	Description

`CartePokemon(tc)`
 	
`CartePokemon(tc,tp,ec,noP,d,niP,ne,pvP,tfp,vfp,tfp,vrp,pr,t)`
	`String`		Constructeur
			
			Constructeur testé dans une classe de test

`getTypePokemon()`
`getExtensionCarte()`
`getNumeroCarte()`
`getNomPokemon()`
`getDescription()`
`getNiveauPokemon()`
`getNomEvoultuon()`
`getPvPokemon()`
`getTypeFaiblessePokemon()`
`getValeurFaiblessePokemon()`
`getTypeResistancePokemon()`
`getValeurResistancePokemon()`
`getPointRetraite()`
`getTalent()`
		`String`
		`String`
		`String`
		`String`
		`String`
		`int`
		`String`
		`int`
		`String`
		`String`
		`String`
		`String`
		`int`
		`Talent`
			Accesseurs

`setTypePokemon(tp)`
`setExtensionCarte(ec)`
`setNumeroCarte(nc)`
`setNomPokemon(noP)`
`setDescription(d)`
`setNiveauPokemon(niP)`
`setNomEvolution(ne)`
`setPvPokemon(pvP)`
`setTypeFaiblessePokemon(tfp)`
`setValeurFaiblessePokemon(vfp)`
`setTypeResistancePokemon(trp)`
`setValeurResistancePokemon(vrp)`
`setPointRetraite(pr)`
`setTalent(t)`
	`String`
	`String`
	`String`
	`String`
	`String`
	`int`
	`String`
	`int`
	`String`
	`String`
	`String`
	`String`
	`int`
	`Talent`
			Modificateurs

`void ajouterAttaque(a)`	`Attaque`		ajout d'une attaque dans le tableau d'attaques
`toString()`		`String`	Entrée sortie : affichage console de la description de la carte Pokémon

2. Classe Attaque

La classe Attaque est appelée par la classe `CartePokemon` et sert à gérer les données d'une attaque de Pokémon.

Elle est définie par plusieurs attributs :

- `nomAttaque` de type **`String`** servant à gérer le nom de l'attaque
- `typeAttaque` de type **`String`** servant à gérer le type de l'attaque
- `nbEnergies` de type **`int`** servant à gérer le nombre d'énergies de l'attaque
- `nbDegatsAttaque` de type **`int`** servant à gérer le nombre de dégâts de l'attaque
- `nbIncolore` de type **`int`** servant à gérer le nombre de cartes incolores.
- `descriptionAttaque` de type **`String`** servant à gérer la description de l'attaque.

Méthodes	Entrée	Sortie	Description

`Attaque()`
 	
`Attaque(na,ta,nbE,nbI,nbD,da)`
			Constructeur
			
			Constructeur testé dans une classe de test

`getNomAttaque()`
`getTypeAttaque()`
`getNbEnergies()`
`getNbIncolore()`
`getNbDegatsAttaque()`
`getNbDegatsAttaque()`
 	
`setNomAttaque(na)`
`setTypeAttaque(ta)`
`setNbEnergies(nbE)`
`setNbIncolore(nbI)`
`setNbDegatsAttaque(nbD)`
`setDescriptionAttaque(da)`
	
	`String`
	`String`
	`int`
	`int`
	`int`
	`String`
		`String`
		`String`
		`int`
		`int`
		`int`
		`String`
			Accesseurs
			
			Modificateurs

`toString()`		`String`	Entrée sortie : affichage console de la description de l'Attaque

3. Classe Talent

La classe Talent est appelée par la classe `CartePokemon` et sert à gérer les données d'un talent de Pokemon.

Elle n'est définie que par deux attributs :

- `nomTalent` de type **`String`** servant à gérer le nom du talent
- `descriptionTalent` de type **`String`** servant à gérer la description du talent

Méthodes	Entrée	Sortie	Description

`Talent()`
 	
`Talent(nt,dt)`
			Constructeur
			
			Constructeur testé dans une classe de test

`getNomTalent()`
`getDescriptionTalent()`
 	
`setNomTalent(nt)`
`setDescriptionTalent(dt)`
			Accesseurs
			
			Modificateurs

`toString()`		`String`	Entrée sortie : affichage console de la description du Talent

4. Classe Deck

La classe Deck sert à gérer le deck (liste des cartes).

Elle est définie par :

- `d` de type **`ArrayListe<Carte>`** servant à gérer le Deck (liste des cartes)
- `cpt` de type **`int`** qui sera incrémenté à chaque ajout de carte dans le deck
- `TAILLE` constante de type **`int`** initialisée à 60 définissant la taille (nombre maximale de Cartes) du Deck

Méthodes	Entrée	Sortie	Description
`Deck()`			Constructeur
`ajouterCarte(c)`	`Carte`		ajout d'une carte dans le Deck, tant qu'on ne dépasse pas 60 cartes
`retirerCarte()`			suppression de la carte en haut de la pile du Deck
`getCarte(i)`	`int`	`Carte`	accès a une carte dans le Deck : méthode utilisée pour balayer le deck dans la fonction sauvegarde
`rechercherCarte()`			recherche d'une carte dans le Deck sur plusieurs critères dépendant du type de Carte : • Carte Énergie : un seul critère possible (type) • Carte dresseur : deux critères possibles (nom du dresseur ou numéro de la Carte) • Carte Pokémon : deux critères possibles (nom du Pokémon ou numéro de la Carte) Les numéros des cartes dans le deck trouvées sont stockés dans une liste d'entiers
`taille()`		`int`	fournit la taille de la liste de carte contenue dans l'objet "Deck" a la méthode sauvegarde
`ajouterCarteInteractive()`			ajout d'une carte interactive : sélection du type de carte puis appel a la fonction ajouterCarte
`melange()`			une fois le deck rempli, il faut mélanger les cartes de manière a ce que la "pioche" se fasse au hasard
`modifierCarte(nc)`	`int`	`Deck avec carte`	modification d'une carte : on remplace la carte d'indice i par une nouvelle carte qu'on crée de manière interactive donc, en

		modifiée	choisissant de nouvelles valeurs pour les attributs dans cette méthode, on peut changer la valeur d'un des attributs mais pas le type de carte
sauvegardeFichier(deck)	Deck	fichier avec le Deck rempli en paramètre	sauvegarde du deck dans un fichier dont on saisit le nom au clavier
viderDeck()	Deck	Deck avec le contenu du fichier dont le nom est saisi au clavier	vidage du Deck
restaureFichier(deck)			restaure le contenu du fichier dont on saisie le nom au clavier dans un deck vide passe en paramètre
toString()		String	Entrée sortie : affichage console du deck

5. Classe Deck

Méthodes	Entrée	Sortie	Description
main	String [] args		Exécution de l'algorithme principal

Cette classe contient l'algorithme principal défini dans le pargraphe I.

CONCLUSION :

Cette phase conception détaillée permet de préparer et faciliter l'écriture du code source.