Projet de Semestre 2

Développement d'un gestionnaire de cartes Hearthstone

Implémentation

18 juin 2018

1 Organisation générale du projet

La semaine 25 est consacrée exclusivement au développement proprement dit et à son évaluation. Vous travaillerez par équipe de 4 à 5 étudiants.

Le travail qui vous est demandé cette semaine est le développement d'une application de bureau permettant de manipuler une collection de cartes HEARTHSTONE :[1] En particulier, elle devra permettre de lister ses cartes et de constituer un "deck[2]".

Plus précisément, l'application à développer devra permettre

— de créer des cartes et de es ajouter à sa collection,

— de supprimer des cartes de sa collection,

— d'afficher/de trier les cartes de sa collection,

— de créer un deck, c'est-à-dire de "sélectionner" certaines cartes de sa collection,

— d'afficher/de trier les cartes d'un deck,

— de sauvegarder sa collection de cartes ainsi que les decks créés,

— etc.

1. https://playhearthstone.com/fr-fr/?utm_source=google_paid et https://fr.wikipedia.org/wiki/Hearthstone
2. https://fr.wikipedia.org/wiki/Deck_(jeu)

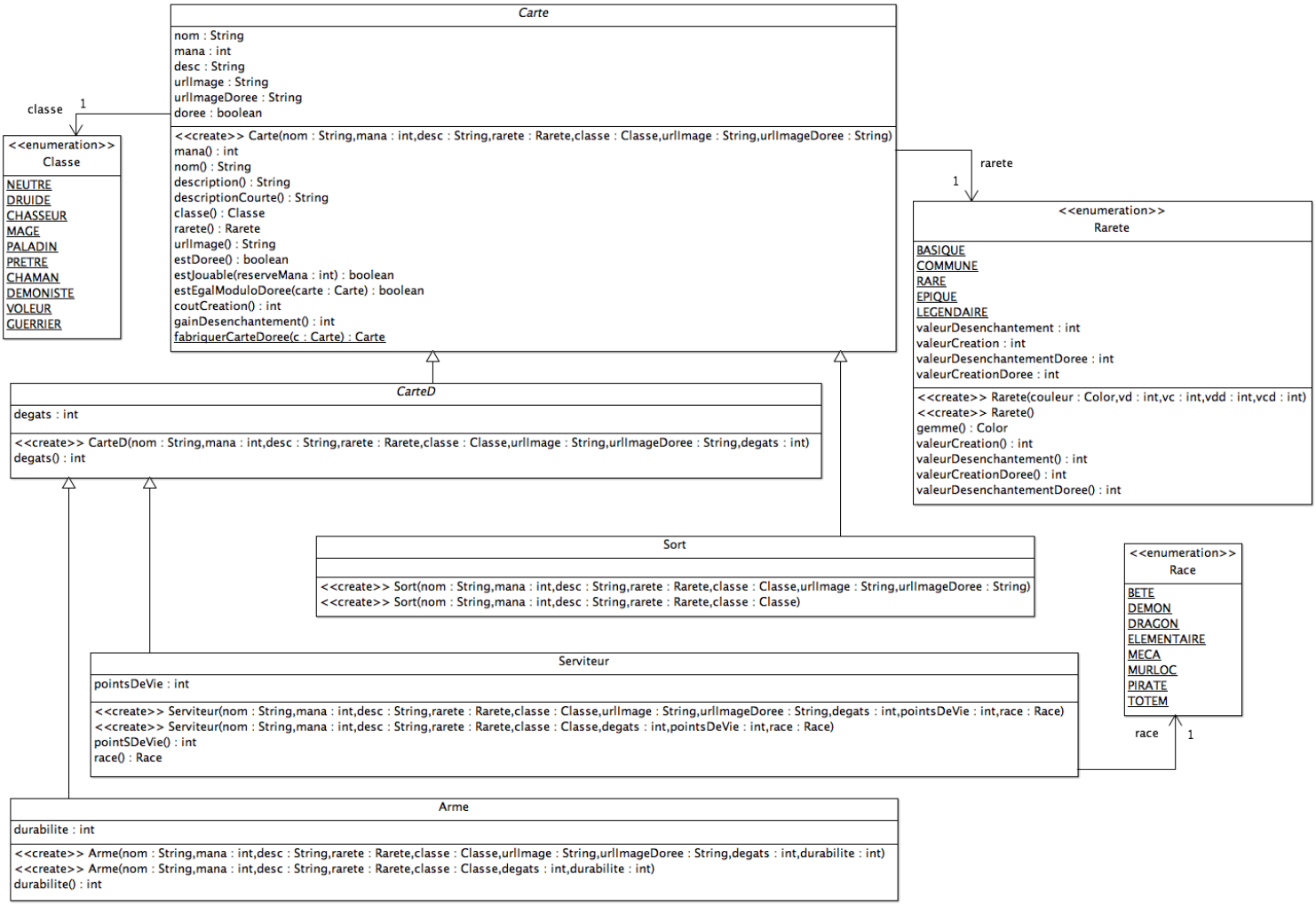

FIGURE 1 – Paquetage `hearthstone.carte`

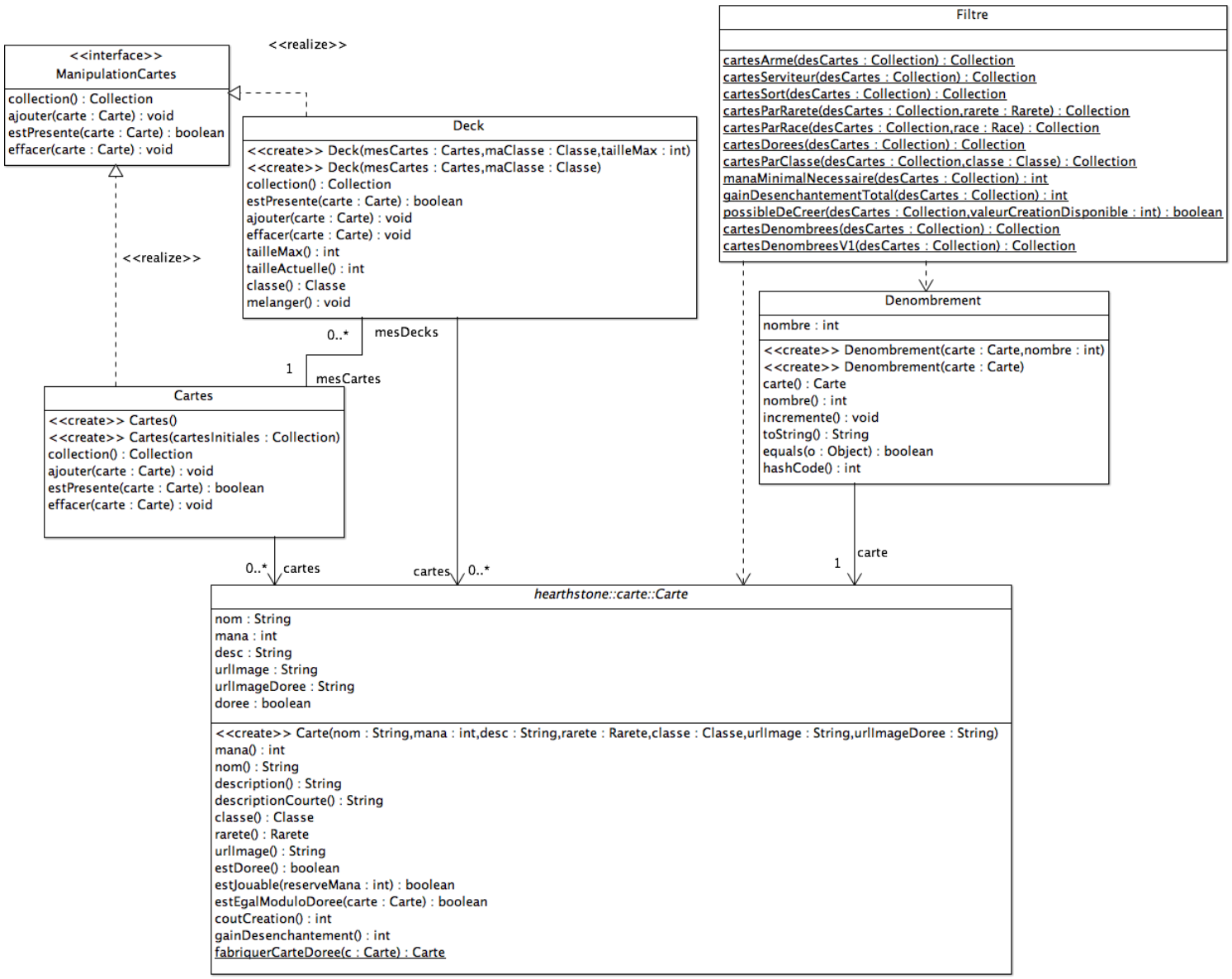

FIGURE 2 – Paquetage `hearthstone.cartes`

2 Travail à réaliser

L'ensemble du développement sera guidé. Il sera impérativement et intégralement fait en Java / Swing / Junit. Plusieurs sous-tâches vous seront demandées, avec un calendrier précis.

Implémentation du modèle de données. Il s'agira de réaliser (une partie de) l'implémentation d'un diagramme de classes concret fourni représentant les données métier que votre application devra manipuler : les deux diagrammes de classes donnés figure 1 et figure 2 décrivent respectivement le contenu du paquetage `hearthstone.carte` et `hearthstone.cartes`.

Concrètement, un ensemble de classes Java vous sera également fourni, via un dépôt Git créé à cette intention ; il s'agira de développer certaines méthodes spécifiquement annotées (`// TODO`). Vous devrez sûrement ajouter des attributs, des méthodes, etc. mais vous ne devrez aucunement modifier la structuration en paquetages, les noms, les visibilités, etc. des classes, des attributs, des méthodes, etc. : votre code doit rester "interfaçable".

Plus précisément, une modélisation des différentes cartes du jeu HEARTHSTONE vous est fournie dans le paquetage `hearthstone.carte`. En particulier,

- Une énumération `Classe` décrit les différentes classes disponibles dans HEARTHSTONE ;
- Une énumération `Race` décrit les différentes races de serviteurs disponibles ;
- Une énumération `Rarete` décrit les différents niveaux de raretés des cartes ;
- Une hiérarchie d'héritage décrit les trois sortes de cartes disponibles :
  - `Carte` décrit l'ensemble des éléments communs à toutes les cartes de HEARTHSTONE ;
  - `CarteD` met en commun l'élément "dégâts" des cartes "Armes" ou "Serviteurs" ;
  - `Arme` décrit une carte "Arme" ;
  - `Serviteur` décrit une carte "Serviteur" ;
  - `Sort` décrit une carte "Sort" ;

  Il est possible de créer une carte dorée à partir d'une carte simple (si sa rareté n'est pas "basique") en utilisant la méthode `fabriquerCarteDoree(Carte c)`.

Dans le paquetage `hearthstone.exception` de nombreuses exceptions levées et/ou à lever sont déjà prédéfinies.

Le paquetage `hearthstone.cartes` contient des classes "utilitaires" qui vous sont fournies :

— Une interface `ManipulationCartes` abstrait les méthodes nécessaires à la manipulation de "paquets" de cartes ;

— La classe `Denombrement` sera utilisée par une méthode de la classe `Filtre` pour dénombrer les exemplaires d'une carte présents dans une collection de cartes donnée ;

— La classe `FabriqueJson` offre différentes méthodes permettant de manipuler des cartes au format JSON. Entre autre, lire des cartes depuis un fichier au format JSON, écrire un fichier JSON à partir d'une collection de cartes ou récupérer des cartes depuis un webservice renvoyant des cartes HEARTHSTONE ;

Le paquetage `hearthstone.cartes` contient également certaines classes qu'il vous faudra implémenter :

— La classe `Cartes` devra décrire l'ensemble des cartes que possède un joueur d'HEARTHSTONE. Attention : on ne souhaite pas ici distinguer les différents exemplaires d'une même carte. Cette classe implémente l'interface `ManipulationCartes` ;

— La classe `Deck` défini un deck, c'est-à-dire un sous-ensemble des cartes possédées par un joueur ; un deck ne contient que des cartes d'une classe définie ou des cartes "neutre". Un deck a une taille maximale fixée. Cette classe implémente également l'interface `ManipulationCartes` ;

Il vous faudra ajouter les attributs nécessaires dans les classes `Cartes` et `Deck`.

— La classe `Filtre` contient certaines méthodes qu'il vous faudra implémenter pour réaliser différents tris/filtres sur l'ensemble des cartes possédées ou bien sur un deck précis. Bien entendu, ces différentes méthodes devront pouvoir être combinées.

ATTENTION : votre code devra respecter des bonnes pratiques de développement, en particulier des nommages explicites, de la documentation du code, etc. Vous justifierez également vos choix d'implémentation, directement dans le code.

Tests du modèle de données implémenté. En parallèle de l'implémentation du modèle de données, il vous faudra écrire des cas de test Junit, permettant de valider chacun des aspects du modèle de données à implémenter.

— vous écrirez des cas de tests Junit pour valider les classes/méthodes fournies [3] ;

— vous écrirez également des cas de tests Junit pour tester les méthodes qu'il vous faut implémenter, afin de valider qu'elles font bien ce qui est attendu qu'elles fassent.

Écrire des tests vous servira à valider votre implémentation au fur-et-à-mesure, mais également des implémentations réalisées par d'autres équipes. Vous remarquerez bien ici l'importance que votre code soit complètement interfaçable et puisse être utilisé par une autre équipe de développement.

ATTENTION : nous aurons également écrit des cas de test de notre coté, et votre implémentation sera également testée par nos soins.

Implémentation d'une IHM. Vous développerez une IHM en Java/Swing qui devra manipuler le modèle de données précédemment implémenté pour réaliser tout ou une partie des fonctionnalités attendues.

ATTENTION : votre application devra scrupuleusement respecter MVC comme vu dans le module IHM.

3 Planning

— *Lundi-mardi :* implémentation du modèle de données + cas de tests

— **Livrable 1 (Mardi, 18 :00) :** livrable du modèle de données implémenté. Un "pull" de votre dépôt Git sera exécuté par nos soins afin de récupérer l'état de votre développement ; faites en sorte que votre code soit fonctionnel.

— *Mercredi :* tests d'autres implémentation du modèle de données, retour de bugs, corrections, etc. A cette intention, vous aurez un accès en lecture au git d'autres projets.

— *Jeudi-vendredi :* développement de l'IHM

— **Livrable 2 (vendredi, 18 :00) :** livrable du projet complet. Votre dépôt Git devra contenir l'ensemble de votre projet, ainsi qu'un fichier README.txt précisant pour votre projet les instructions de compilation/exécution depuis un terminal.

3. N'hésitez pas (le cas échéant) à nous remonter des erreurs dans le code qui vous a été fourni

4 Évaluation

Tous les aspects du projet seront évalués. Entre autre,

— l'implémentation correcte du modèle de données fourni ;

— la justification des choix d'implémentations ;

— la qualité et la pertinence des tests réalisés, ainsi que les rapports de bugs fournis aux autres groupes ;

— le développement d'une IHM ;

— l'implication et le comportement dans le déroulement du projet