Rapport projet Tarot

I) Introduction

L'objectif de ce rapport est de présenter les choix architecturaux du projet, sa structure, les différentes étapes suivies et les problèmes rencontrées.

II) Analyse et modélisation

II.1) Analyse

Une carte possède un numéro, un nom, une description et une image. Le numéro et le nom sont obligatoires pour que la carte puisse être crée, la description et l'image sont facultatifs.

Le tarot divinatoire possède 22 arcanes majeurs. Il faut donc regrouper les cartes dans une liste qui formera un paquet de cartes, sur lequel plusieurs opérations pourront se faire. Comme la carte « Le Mat » ne possède pas de numéro dans le tarot je lui ai attribué le numéro 0 afin de garder l'ordre des cartes et que leur indice dans la liste corresponde à leur numéro.

J'ai décidé de travailler seulement avec les arcanes majeurs en me basant sur le tarot divinatoire de Marseille qui n'est basé que sur les arcanes majeurs. Ainsi quand on parle de « carte » il s'agira toujours d'un arcane majeur.

On applique la méthode orientée objet. Pour cela on applique l'abstraction au domaine du problème pour identifier les objets, puis on applique l'abstraction aux objets pour identifier leur état et leur comportement.

Tableau 1 présente les objets du problème, leur état et leur comportement

Objet	Etat	Comportement
Carte	Numéro, nom	Interprétation
Paquet	Liste de cartes	Gérer les cartes
Données	Constantes	Contenir les constantes

Tableau 1 - Objets du problème

On crée la classe Card abstraction de l'objet carte. On crée la classe Deck abstraction de l'objet paquet. On crée la classe Data abstraction de l'objet données.

Chaque classe contiendra des getters pour récupérer les attributs et des setters pour les modifier qui ne sont spécifiés dans le schéma.

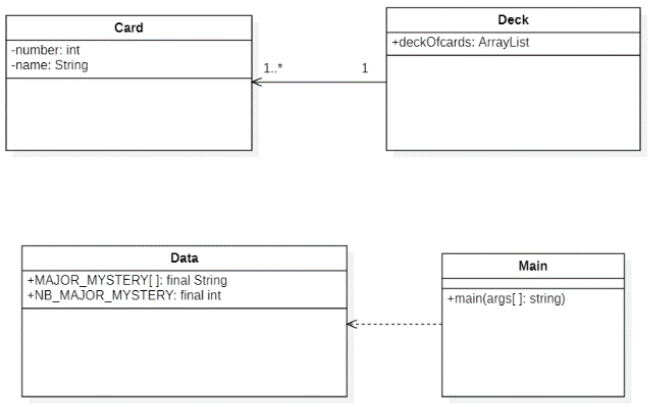

Figure 1 - Diagramme de classe modélisant les objets du problème

L'application proposera un menu permettant d'utiliser les fonctions suivantes :

- Afficher une carte et ses propriétés
- Créer une nouvelle carte
- Modifier une carte existante
- Supprimer une carte
- Afficher le paquet de carte
- Rechercher une carte parmi un paquet de cartes
- Gérer l'application

Chaque fonction sera expliquée et détaillée avec des diagrammes dans les parties suivantes.

II.2) Choix de modélisation

J'ai choisi une modélisation MVC (Modèle – Vue – Contrôleur) afin de séparer les données (le modèle) de leur représentation graphique (la vue). Ainsi les données peuvent être réorganisées sans que cela n'affecte leur représentation ; de même les modifications appliquées aux représentations graphiques n'ont aucun impact sur l'organisation des données. Le contrôleur, lui, s'occupera de gérer la synchronisation entre les données et la représentation graphique.

III) Système de carte basique

Le système de carte basique permet de créer des cartes avec numéro et un nom et contient une méthode permettant d'ajouter une description à la carte. Ainsi après avoir créé tous les arcanes majeurs ont pourra les ajouter dans une liste ce qui formera le paquet de cartes.

Il est possible de supprimer une carte du paquet avec l'indice de la carte.

IV) Extension et recherche

Pour étendre le projet, j'ai ajouté une fonction de modification de carte qui permet changer le numéro et le nom de la carte, a spécifié en paramètre de la fonction. Une carte possède une image, donc j'ai utilisé une méthode pour ajouter une image de type ImageIcon à la carte. Chaque carte possède donc sa propre image. Je récupère les images d'un répertoire « images » et je les associe à la carte correspondante. Par exemple la carte « Le Bateleur » aura l'image du bateleur ou son numéro et son nom sont affichés.

Pour consulter la collection de cartes il suffit de récupérer la liste des cartes et de les afficher.

J'ai également implémenté une fonction de recherche de carte basé sur deux critères : le numéro et le nom. Je parcours la liste de cartes et compare le numéro et le nom de chacune des cartes du paquet et je le compare avec ceux de la carte recherchée. Si une carte du paquet possède le numéro et le nom spécifié alors elle existe et est affiché sinon rien n'est affiché.

V) Sauvegarde

Le système de sauvegarde implémente l'interface *Serializable* afin de permettre à l'utilisateur de sauvegarder ses cartes. On crée flux de sortie vers le fichier de sérialisation en utilisant la classe **FileOutputStream**. Puis, il faut créer un flux objet de la classe **ObjectOutputStream** qui permet de sérialiser un objet grâce à la méthode writeObject() qui écrit l'objet dans le flux de sortie c'est-à-dire le fichier. La désérialisation est la procédure inverse et permet de récupérer les objets en lisant le fichier.

VI) Interface graphique

J'ai implémenté plusieurs classes qui s'occupe chacune de gérer une fonction de l'application et

- MyFrame : Fenêtre principale qui contient le menu et le message de bienvenue. Le menu est composé comme suit :
    - Un item carte avec les sous items afficher carte, créer carte et modifier carte
    - Un item paquet avec les sous items afficher paquet, rechercher carte et supprimer carte
    - Un item gestion de l'application
    - Un item quitter
- PanelCards : Panel principal qui contient les panels de chaque item du menu et gère leur affichage grâce à l'utilisation d'un CardLayout, un gestionnaire permettant de superposer le contenu. Les fonctions de suppression et quitter sont gérer dans des boites de dialogues pour simplifier. Le clic sur le bouton « OK » de la boite de dialogue exécute la fonction correspondante.

- PanelDeck : Il s'agit du panel qui affiche le paquet de cartes. Ce panel est basé sur un gestionnaire CardLayout et possède quatre boutons :
    - « << » pour afficher la première carte du paquet
    - « < » pour afficher la carte précédente à la carte courante
    - « > » pour afficher la carte suivante à la carte courante
    - « >> » pour afficher la dernière carte du paquet.

  Pour gérer les boutons il faut réinitialiser les indices pour pouvoir « bouclé » et ne pas avoir d'erreur de dépassement d'indice du paquet de cartes.

- PanelDisplayCard : Ce panel permet d'afficher une carte et ses propriétés en utilisant un BorderLayout, gestionnaire qui arrange les composants en 5 régions : nord, sud, est, ouest et centre. Dans ce cas je n'ai utilisé que les zones nord, centre et sud. La zone nord contient une liste déroulante contenant le nom des cartes existantes et un bouton sur lequel il faut cliquer pour afficher la carte souhaitée. La zone centre contient l'image de la carte et utilise un CardLayout. La zone sud contient un label avec la description de la carte affichée.

- PanelForm : C'est un formulaire pour créer une nouvelle carte utilisant un GridBagLayout, gestionnaire qui dispose les composants selon des contraintes sur les axes x et y du panel. Ainsi le panel possède 3 zones de saisie pour le numéro, le nom et la description de la carte, un bouton renvoyant à un explorateur de fichiers de l'ordinateur pour ajouter une image et un bouton pour que les données du formulaire soient récupérées et ajoute une carte.

- PanelFormSearch : Il s'agit d'un formulaire pour rechercher une carte utilisant un GridBagLayout. Ainsi le panel possède 2 zones de saisie pour le numéro et le nom de la carte et un bouton pour que les données du formulaire soient récupérées et lance la fonction de recherche de la carte, qui affiche l'image de la carte si celle-ci existe.

- PanelManagement : Gestion de l'application qui permet de changer la couleur et la police de l'application.

- PanelSearch : C'est le panel principal de recherche, utilisant un BorderLayout, qui permettra d'afficher la carte rechercher au centre et contient le formulaire de recherche à droite.

- PanelUpdateCard : Ce panel est un formulaire de pour modification d'une carte utilisant un GridBagLayout. Ainsi le panel possède, une liste déroulante pour choisir la carte à modifier, 2 zones de saisie pour le numéro et le nom modifiés de la carte et un bouton pour que les données du formulaire soient récupérées et lance la fonction de modification de la carte.

Avec ces nouvelles implémentations on obtient le diagramme de classe suivant (Figure 2) :

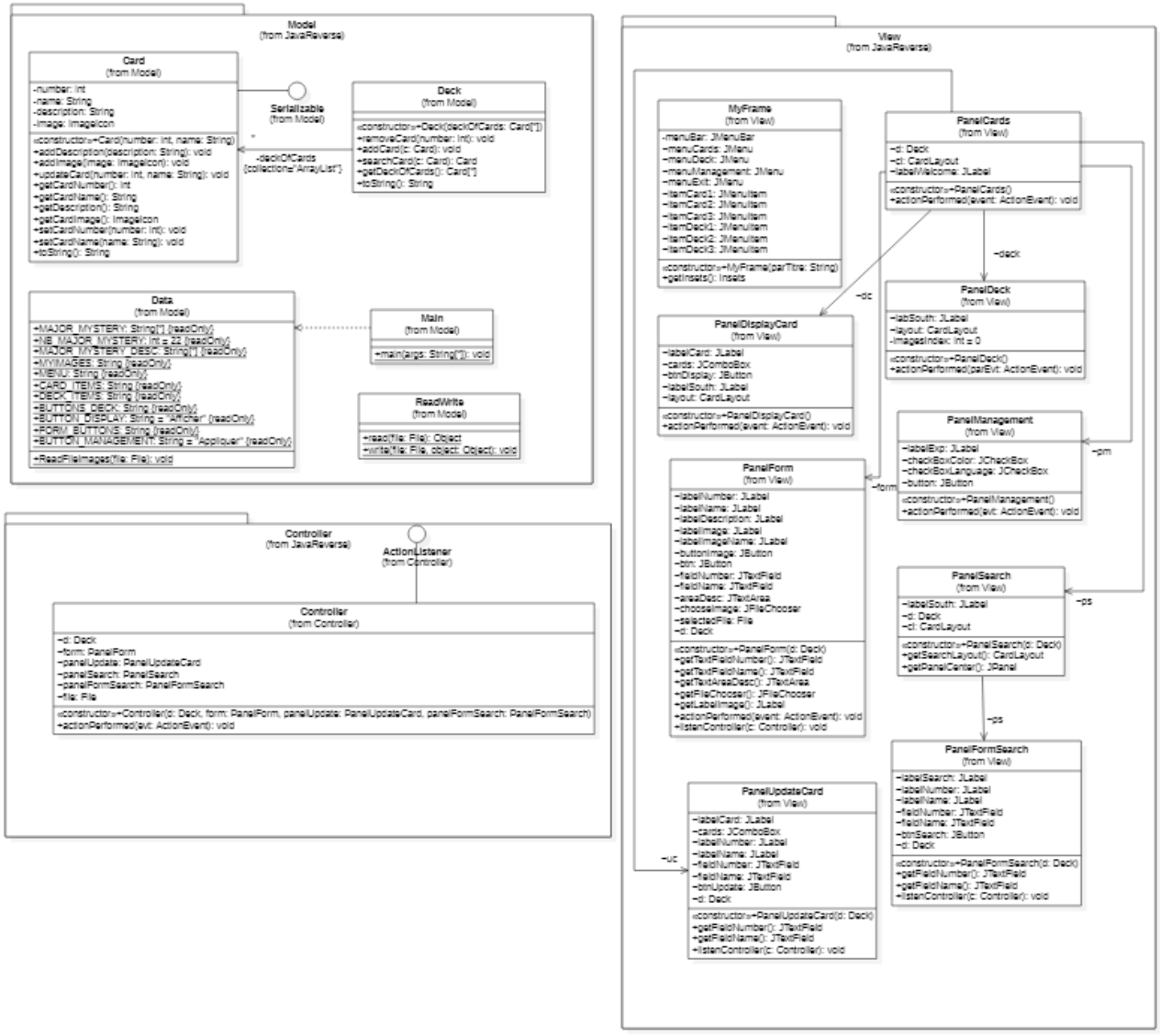

Figure 2 - Diagramme de classe modélisant l'application du tarot

VII) Lecture du futur

J'ai implémenté une méthode qui permet de mélanger le paquet de demander à l'utilisateur de choisir 4 cartes puis renvoie l'interprétation en fonction des cartes choisies. A chaque numéro de carte entre par l'utilisateur, le programme vérifie que le numéro ne dépasse pas la taille de la liste et réduit la taille de la liste à chaque carte choisie.

Je n’ai pas eu le temps d’implémenter la représentation graphique de la lecture du futur, elle n’est donc disponible seulement en utilisant la console.

VIII) Tests

Deux classes de test ont été implémentées en utilisant JUnit pour tester les méthodes des classes Card et Deck. Tous les tests ont réussi.

IX) Problèmes rencontrés

J’ai rencontré un problème avec les images des cartes car quand je les avais ajoutés je ne m’étais pas rendu compte qu’elles étaient intégrées dans le désordre et de plus, j’avais merge la branche où j’avais ajouté les images avec la branche principale. J’ai donc dû utiliser un fichier contenant le nom des images. Après lecture du fichier, chaque ligne du fichier était insérée dans un tableau afin que les images soient dans le bon ordre. Je n’avais ensuite plus qu’à utiliser le tableau pour avoir le nom des images et les afficher dans l’ordre.

Je n’ai pas pu implémenter la sérialisation JSON suite à une erreur qui n’a pas pu être résolue. En effet, il semblerait que JSON ne permette pas de sérialiser les images se qui devait être contourné d’une autre façon que je n’ai malheureusement pas eu le temps d’implémenter.

X) Conclusion

Dans l’ensemble le projet est abouti. La lecture du futur aurait pu être implémentée sous forme de représentation graphique pour étendre le projet mais n’a pas pu être réalisé par manque de temps.