Sylvain Archenault
Yves Houpert

Projet Informatique :

Langage Java :

Le Jeu de Uno

Projet GM3
Juin2005

Table des matières

Chapitre 1

Introduction

Notre projet de Java est de réaliser **un jeu de UNO**. Ce jeu sera simplifié par rapport aux règles du jeu normal, car il doit répondre aux contraintes du sujet. En effet, le but de ce projet est de mettre en exergue **les principes fondamentaux de la programmation objet**, c'est à dire **l'héritage** entre les classes, la gestion des **interfaces** ou des **classes abstraites**,etc.

Dans une première partie, nous rappellerons brièvement les règles de base du UNO.

Pour bien répondre au problème, nous nous inspirerons **du cycle général de la conception objet** que nous avons vu en cours. Cette méthode de conception nous permettra d'avoir une présentation claire et structurée de notre projet.

Dans une autre partie, nous parlerons plus précisément du **polymorphisme**, notion propre au langage JAVA . Nous avons essayé de l'utiliser optimalement et nous exposerons comment.

Enfin, pour terminer nous parlerons des différents problèmes que nous avons rencontré lors de la réalisation du projet et comment nous y avons remédié.

Chapitre 2

Règles du jeu.

Rappelons que la règle a été simplifiée par rapport au jeu d'origine pour bien répondre aux demandes du projet.

2.1 But du jeu

Le but du jeu est d'être le premier joueur à se débarrasser des 7 cartes qui lui auront été distribuées.

2.2 Contenu du jeu

108 cartes réparties de la façon suivante :
19 cartes bleues, numérotées entre 0 et 9.
19 cartes rouges, numérotées entre 0 et 9.
19 cartes jaunes, numérotées entre 0 et 9.
19 cartes vertes, numérotées entre 0 et 9.
8 cartes "+2", (2 de chaque couleur).
4 cartes "JOKER".
4 cartes "+4".

2.3 Préparation du jeu

1. Mélanger soigneusement les cartes et en distribuer 7 à chaque joueur.
2. Faire une pile avec le reste des cartes, faces cachées : elle constituera LA PIOCHE.
3. Retourner la première carte de la pioche, pour la poser face visible à côté de la pioche, afin de commencer LE TALON. Cette carte servira comme base de départ.

2.4 Déroulement de la partie

Le joueur 1 doit recouvrir la carte supérieure du talon par une carte de la même couleur, ou portant le même numéro ou le même symbole que celle-ci. Par exemple, si la carte supérieure du talon est un 7 rouge, il peut jouer n'importe quelle carte rouge ou un 7 de n'importe quelle couleur. Il peut également jouer une carte Joker.

Si le joueur ne possède pas de cartes lui offrant une de ces possibilités, il doit alors tirer la carte supérieure de la pioche. Si cette carte peut être jouée, il peut alors la jouer, en la posant sur le talon sinon il passe son tour, en conservant bien évidemment cette carte supplémentaire.

A son tour, il est permis de ne pas jouer, même si l'on possède une carte jouable. Il faut alors tirer une carte de la pioche. Si elle est "jouable", celle-ci peut alors être jouée immédiatement.

Le joueur qui s'est débarrassé le premier de toutes ses cartes est le vainqueur du jeu.

2.5 Conventions

2.5.1 Le talon

La première carte de la pioche est posée face visible à côté de la pioche, afin de commencer le talon. Cette carte sert de base de départ. Si c'est une carte spéciale, certaines conventions particulières sont applicables (voir chapitre suivant).

2.5.2 Les cartes spéciales

Voici leur signification et les effets qu'elles produisent.

1. "+2"

Lorsque cette carte est jouée, dans tous les cas le joueur suivant doit tirer 2 cartes dans la pioche et passe son tour. Idem si c'est la première carte du talon. Une telle carte ne peut être jouée que sur une carte de la même couleur ou sur une autre carte "+2".

1. "JOKER" (4 couleurs)

Le joueur qui dépose cette carte peut choisir de changer la couleur (il annoncera son choix en jouant la carte) ou de continuer dans la couleur demandée, peu importe le numéro. Si c'est la première carte du talon, le donneur choisira alors la couleur de départ. Une telle carte peut être jouée après n'importe quelle autre carte.

1. "+4"

C'est une carte bien particulière : elle permet de pénaliser le joueur suivant tout en remplissant les fonctions de Joker. Lorsque cette carte est jouée, le joueur suivant doit tirer 4 cartes dans la pioche .

2.6 Fin d'une manche

Le premier joueur à s'être débarrassé de toutes ses cartes remporte la partie.

Voici donc les règles principales de notre UNO personnalisé.

Chapitre 3

Le cycle général de la conception par objet.

Nous présenterons notre projet en respectant les consignes du cours c'est à dire en tirant profit des 5 grandes parties du cycle :

1. **Identification des entités du problème à traiter.**
2. **Structuration du problème.**
3. **Identification des opérations.**
4. **Implémentations des opérations.**
5. **Lancement de l'application.**

3.1 Identification des entités du problème à traiter.

Les classes que nous proposons sont les suivantes :

Nous avons obtenu les attributs de chaque classe en réfléchissant à ce qui caractérisait précisément celle-ci (par exemple une carte est caractérisée par sa couleur et son numéro en général. Cette identification a été réalisée à l'aide du logiciel *Umbrello*.

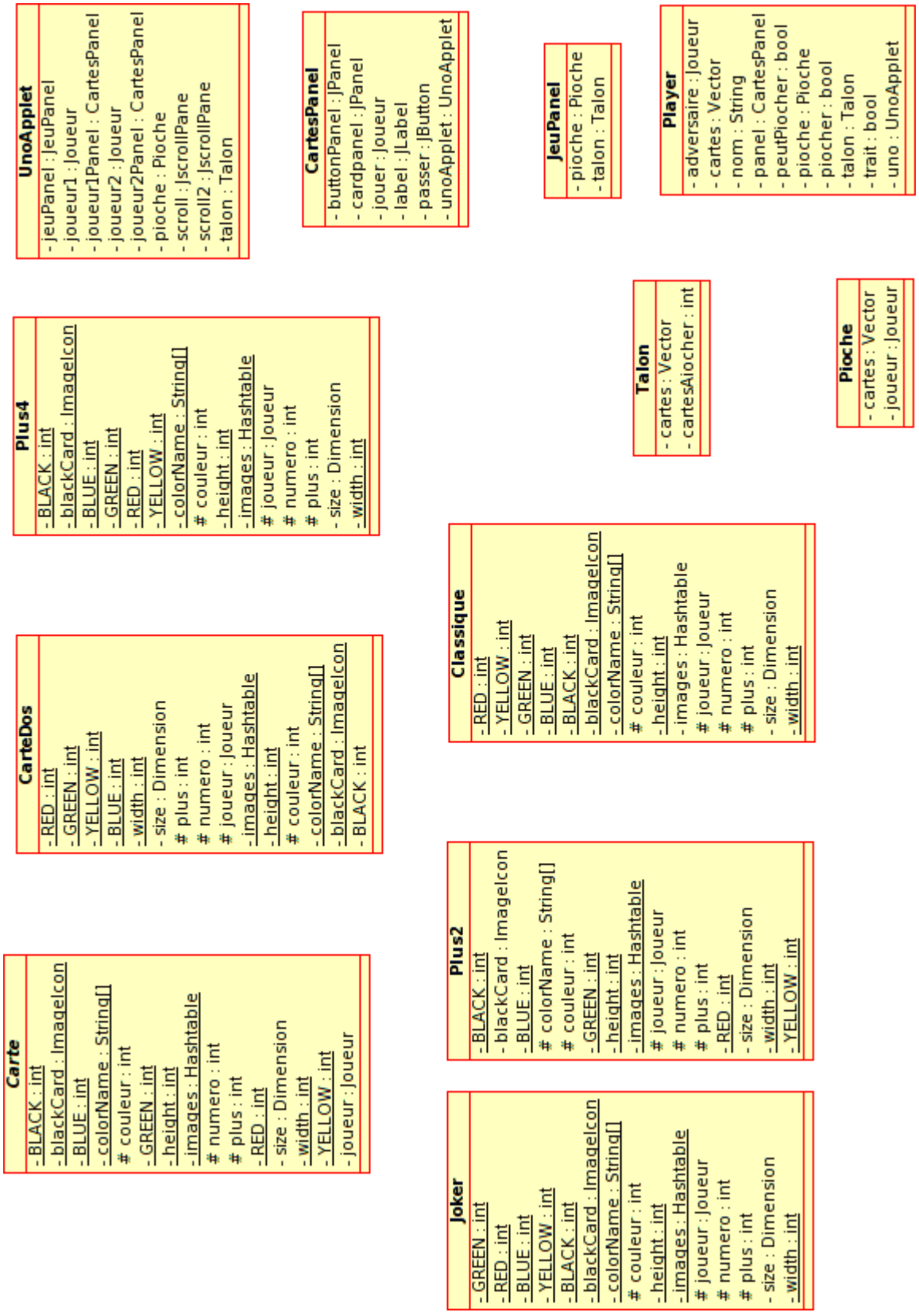

FIG. 3.1 – Étape 1 du cycle de conception

3.2 Structuration du problème.

Nous avons ensuite pensé à structurer le problème. En effet, il nous est apparu pratique d'utiliser une classe abstraite Carte. On peut créer des objets Classique,Joker, Plus2, Plus4 qui sont les différents types de cartes du jeu. Ce sont donc des sous classes de Carte. La notion d'héritage va donc être traitée dans ce mini projet. De plus, nous avons créé une Applet dans laquelle le jeu se déroulera.

Voici dons les relations entre les différentes classes.

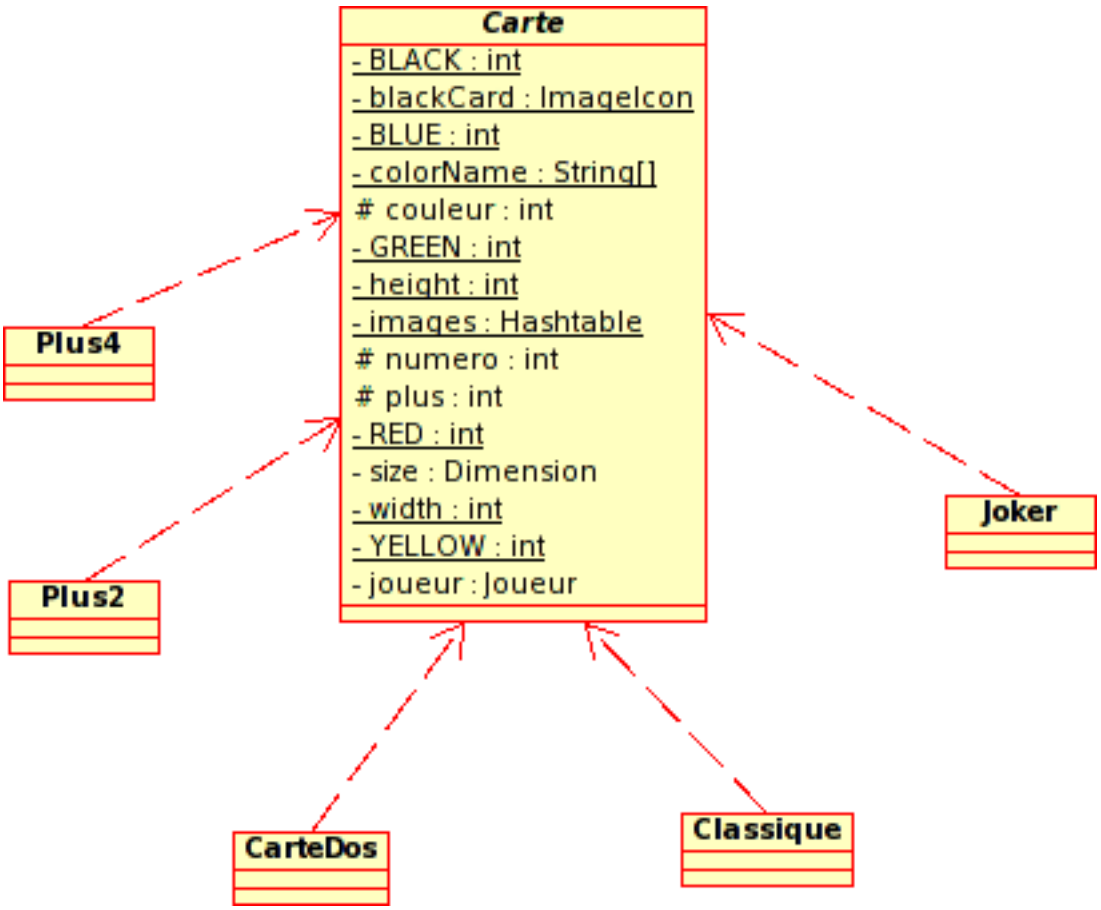

FIG. 3.2 – Les classes Carte et ses sous-classes

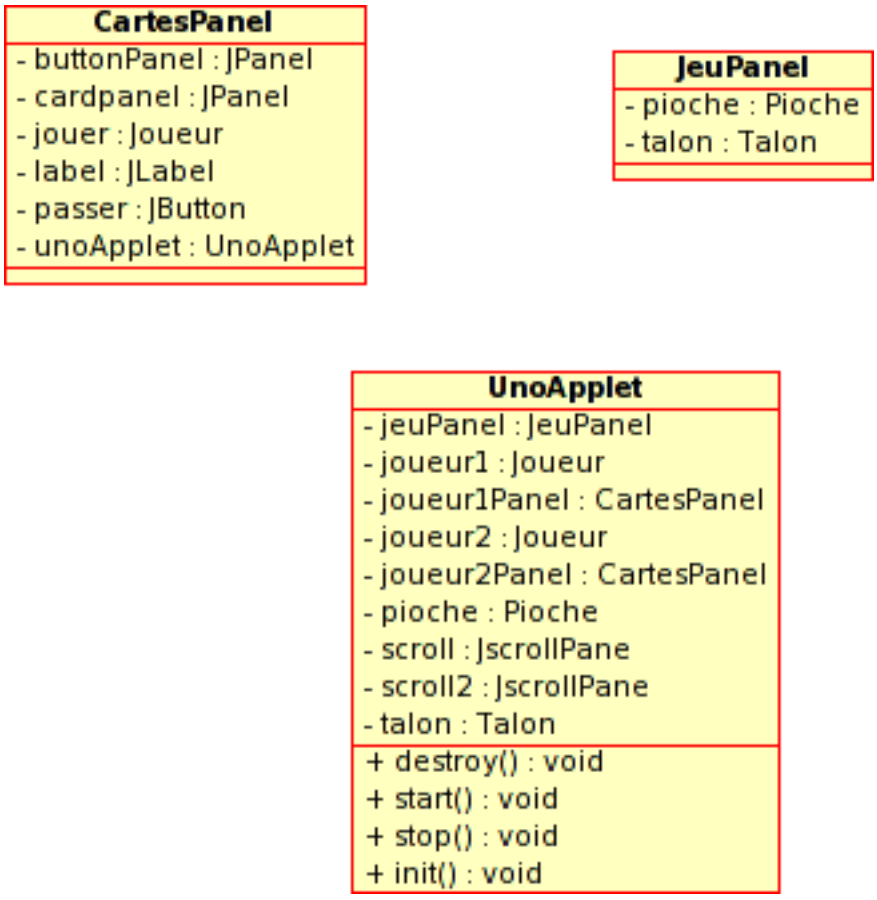

FIG. 3.3 – Les classes graphiques

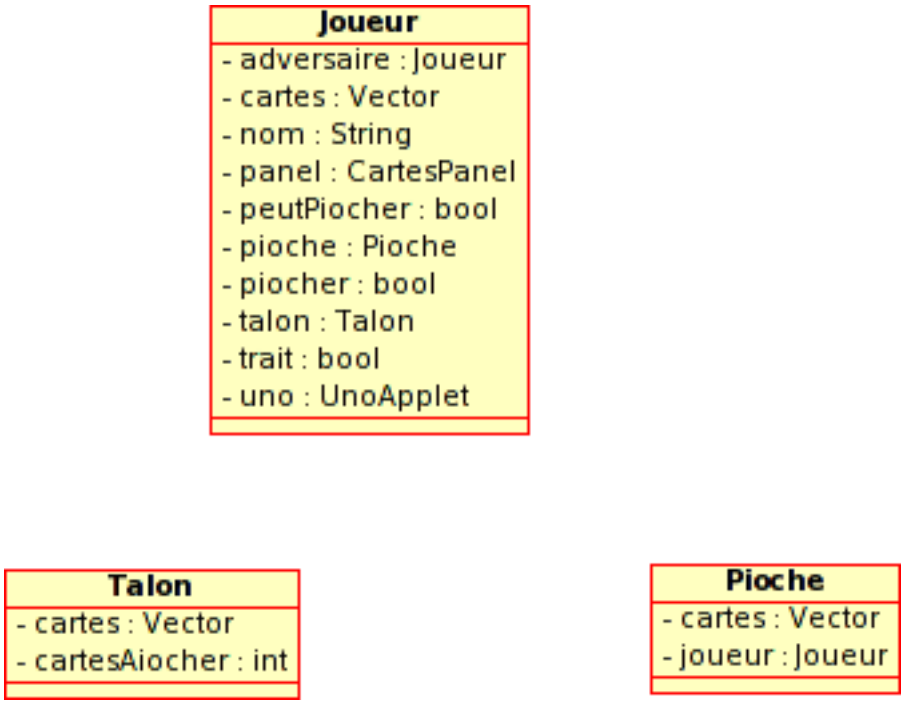

FIG. 3.4 – Les autres classes

On voit donc bien l'aspect objet du projet. On voit bien apparaître les méthodes *init*(),*destroy*(),*stop*() et *start*(), obligatoires au fonctionnement d'un applet en Java. Nous trouverons les autres méthodes dans la suite de l'étude.

3.3 Identification des opérations.

Nous avons trouvé les méthodes à implanter grâce à l'utilisation de scénarios. Nous avons simulé chaque étape du jeu et les différents cas que pouvaient causer l'utilisation de cartes spéciales. Nous obtenons de nombreuses méthodes. Des méthodes abstraites bien sûr comme la méthode *MouseClicked* par exemple. Cette méthode est définie comme méthode abstraite dans la classe Carte, mais elle n'a pas les mêmes caractéristiques selon sur le type de carte (Classique, Joker, etc) sur lequel on clique.

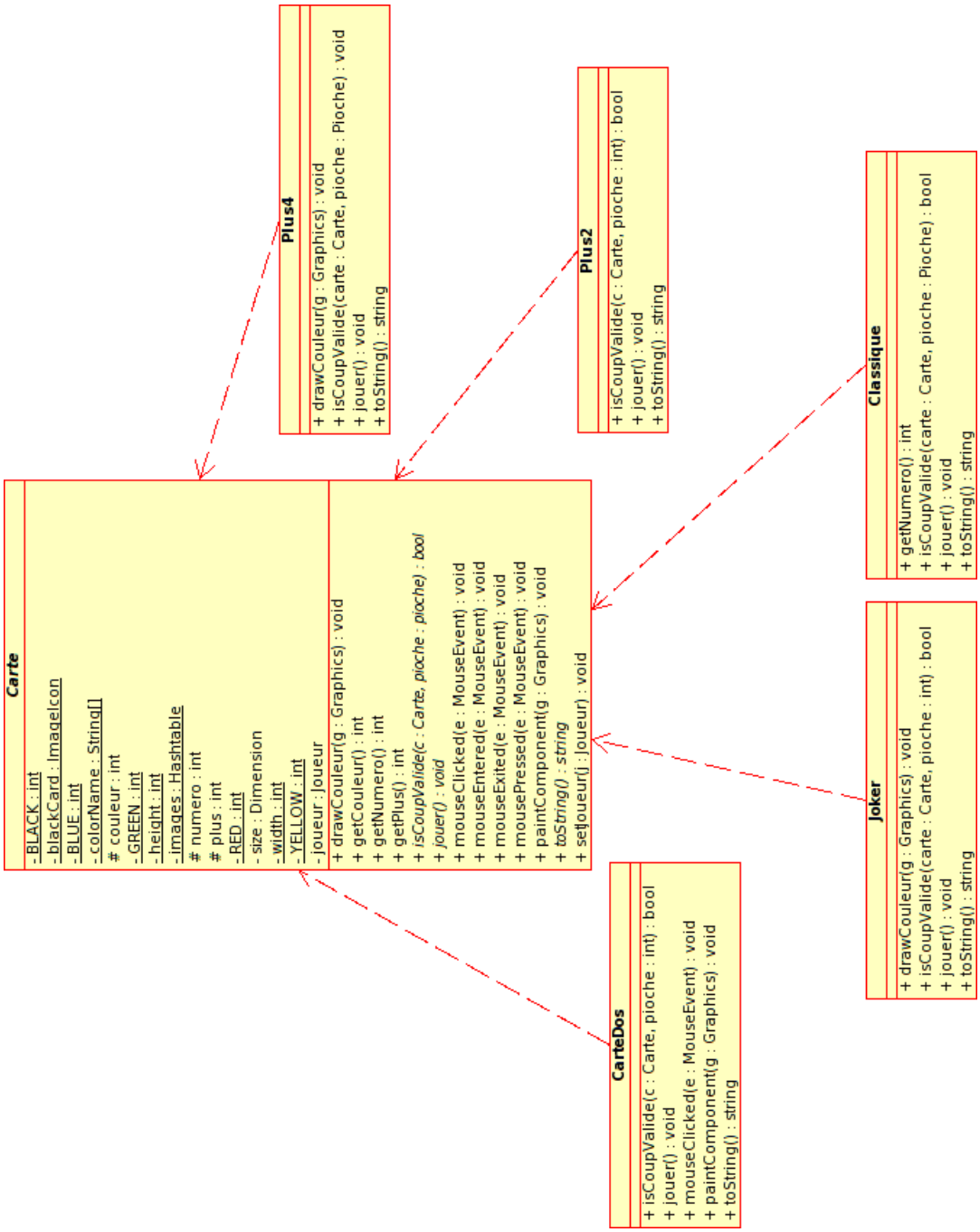

FIG. 3.5 – Diagramme de classes - Classe Carte et ses sous-classes

On voit bien ici les méthodes abstraites. La méthode isCoupValide() est la méthode qui vérifie si le coup est valide. Elle est différente pour chaque sous classe puisque il n'y a pas les mêmes règles de jeu pour chaque type de carte, donc elle est redéfinie dans chaque classe héritant de la classe Carte. il en est de même par exemple pour la méthode toString().

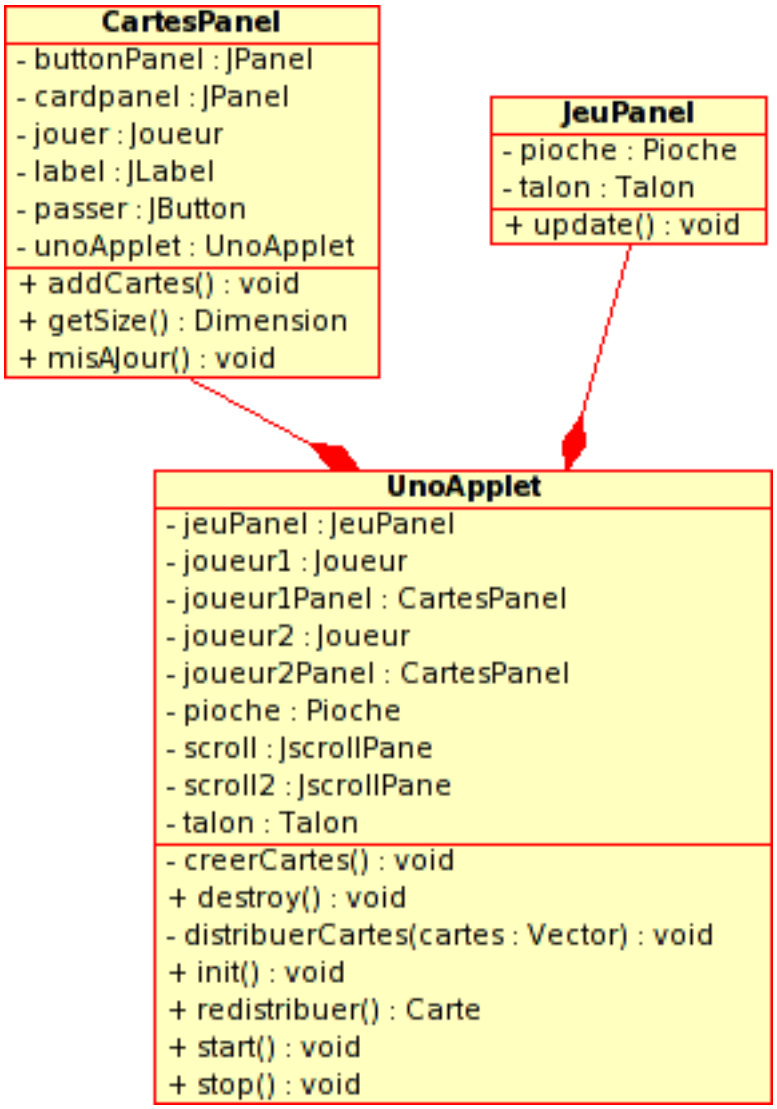

FIG. 3.6 – Diagramme de classes - Les classes “graphiques”

On a également rajouté en plus des méthodes une relation de composition entre la classe *UnoApplet* qui est composée d'un *CartesPanel* qui affiche les cartes du joueur et d'un *JeuPanel* qui affiche le talon et la pioche.

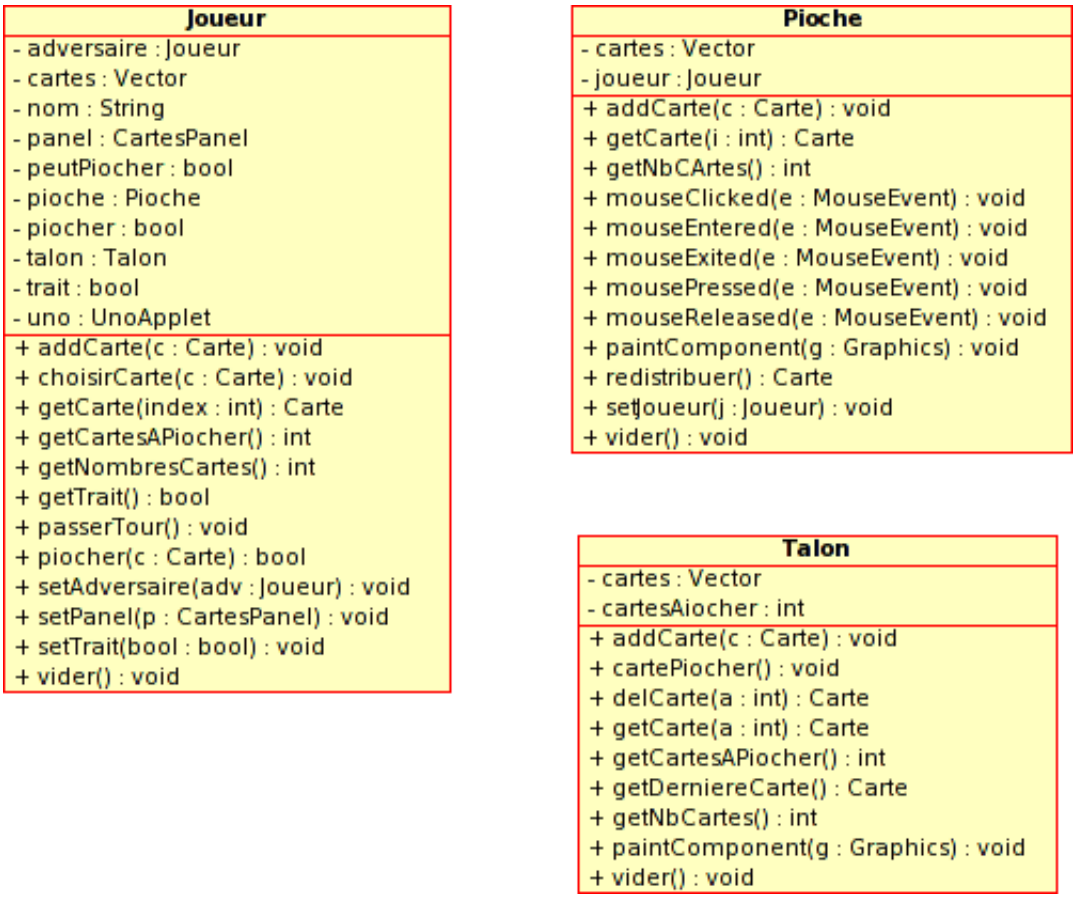

FIG. 3.7 – Diagramme de classes - Les autres classes

La classe joueur comprend également de nombreuses méthodes. En effet, on a besoin de savoir qui a le trait. Nous avons opté pour une méthode nous renvoyant le trait du joueur.

Nous avons également besoin de savoir combien le joueur doit piocher de cartes ou de savoir quelle est la dernière carte jouée. Nous avons donc programmé les méthodes *getDernièreCarte*() et *getCartesAPiocher*() dans la classe talon.

3.4 Implémentation des opérations.

Il est très long de développer cette partie. Nous avons donc généré une documentation avec le rôle et la définition de chaque méthode à *http ://gm.insa-rouen.fr/~archenas/projets/uno/doc/*. Nous n'avons pas désiré ajouté cette documentation au rapport car il est beaucoup plus pratique de la consulter via internet.

3.5 Lancement de l'application.

Pour lancer l'application, nous allons d'abord créer deux joueurs auxquels nous allons associer deux *CartesPanels* où seront affichés leurs cartes. Nous créons aussi un talon et une pioche auxquels on associe un *JeuPanel* où ils seront affichés.

Nous devons ensuite créer les cartes et les distribuer, c'est le rôle de la méthode creerCarte de *UnoApplet*.

Le jeu peut commencer.

Chapitre 4

Les concepts de la programmation objet.

4.1 Le polymorphisme

Dans cette section, nous allons mettre en exergue deux exemples dans lesquelles le polymorphisme intervient.

4.1.1 La méthode *isCoupValide(Carte c)*

Cette méthode abstraite de la classe Carte a pour but de savoir si un coup est valide. Elle renvoie le résultat sous forme d'un booléen. Cette méthode prend en paramètre la carte qui se trouve sur le talon.

Cette méthode est donc différente suivante le type de carte (Classique, Plus2, etc). Nous avons donc programmé une méthode différente dans chaque sous-classe, en implémentant les règles propres au type de carte.

Grâce au polymorphisme, la programmation du déroulement du jeu est grandement facilitée. Expliquons cela.

Tout d'abord, comme on peut le remarquer dans le diagramme des classes : , la liste des cartes jouées est stockée dans un vecteur. Grâce au polymorphisme, on ne stocke pas des cartes Classique, Plus2, Plus4 ou bien Joker, mais des Cartes. Par conséquent, lorsqu'on appelle la méthode *isCoupValide(Carte c)*, c'est *Java* qui appelle la méthode de la classe approprié.

Par conséquent, on obtient le résultat souhaité, sans avoir des tests conditionnels imbriqués, ce qui aurait rendu la programmation beaucoup plus compliquée.

4.1.2 La méthode *toString()*

Comme nous avons vu en cours de Java cette année, la redéfinition de cette méthode offre un exemple classique de polymorphisme, elle est de plus très pratique. Voyons comment nous avons utilisé cette méthode dans notre programme.

Tout d'abord, nous devons expliquer comment nous avons stocké les images relatives à l'interface graphique. Les images du jeu sont stockées dans une table de hachage. Dans un table de hachage chaque élément est identifié par une nombre comme dans un tableau mais aussi une chaîne de caractère. Nous avons donc pour chaque image associé une chaîne de caractère.

Exemple : L'image de la carte bleu portant le numéro 5 est représentée par la chaîne Bleu5

Dans chaque sous-classes, la méthode *toString()* renvoie la chaîne de caractère qui identifie la carte dans la table de hachage.

Exemple : Dans la classe Classique, la méthode toString renvoie une chaîne de caractère de la forme CouleurNuméro.

Nous arrivons à la phase de dessin. Pour dessiner les cartes, nous devons choisir la bonne image. Grâce à la méthode *toString()*, nous n'avons plus besoin de passer par des tests. Il suffit de l'appeler (Java choisit automatiquement la méthode *toString* de l'objet correspondant), et nous obtenons la chaîne de caractère de la carte, il reste plus qu'à récupérer l'image stockée dans la table de hachage.

Ceci offre donc une très grande souplesse de programmation.

4.2 Les interfaces

Nous n'avons pas rencontré le besoin de créer nos propres interfaces. Néanmoins, nous avons tout de même utilisé une interface fournie par Java : *MouseListener*.

Cette interface permet de gérer les événements de la souris, notamment les cliques. En effet, nous devions savoir quand l'utilisateur clique sur une carte. De plus, la réaction est différente si l'utilisateur clique sur une carte retournée ou sur une carte face cachée.

Par conséquent nous avons défini dans la classe *Carte* l'action quand le joueur clique sur une carte retournée. Nous avons redéfini cette méthode dans la classe *CarteDos*, puisque le comportement est différent. Nous voyons encore ici un des bienfaits de la programmation objet, puisque grâce à l'héritage, la différence de traitement se fait très naturellement.

Chapitre 5

Conclusion.

Ce jeu de UNO en langage Java fut très intéressant à concevoir et à programmer.

Ce projet nous a permis de bien comprendre **les qualités de la programmation objet**. Nous avons essayé d'utiliser au mieux ses caractéristiques tels que **l'héritage ou les classes abstraites.**

Les propriétés de la programmation sont très pratiques et permettent une **bien meilleure conception et réalisation du projet**.

Il faut également préciser que **l'apprentissage du cycle général de conception par objet fût très intéressant et a permis une réalisation optimale.**

Au début du projet, nous avons trouvé les différentes classes et méthodes à créer grâce à **l'utilisation de scénarios et de diagrammes de classes**. Cette démarche permet d'avoir une réflexion structurée tout au long de la programmation.

Il aurait été très intéressant de pouvoir mettre le jeu en réseau. Nous n'avons pas eu le temps de le faire mais cette amélioration pourrait faire l'objet d'un mini projet futur.

Table des figures