Etude du diagramme de classes :

Le diagramme de classes quant à lui, nous renseigne sur la structure interne du système. Il permet notamment de visualiser les interactions entre les objets du système pour réaliser les cas d’utilisation. Nous pourrons donc voir les relations entre les classes, lesquelles se constituant d’attributs, de méthodes et d’interface.

La classe centrale du système est la classe Partie qui contiendra la méthode main () spécifiant où doit débuter l’exécution du programme. La classe Partie contient des informations sur le nombre de joueurs, joueurActif quel joueur est en train de jouer. nbTours sert à connaitre le nombre de tours joué, un tour est effectué lorsque tous les joueurs ont posé une ou plusieurs cartes. La méthode incrementerJoueurActif() est utilisé après que chaque joueur est joué son coup et permet de connaitre qui est en train de jouer.

Une Partie est composée de joueur. La classe Physique et Virtuel héritent de Joueur. Les classes Defensif et Offensif héritent de la classe Virtuel et utilisent une stratégie. Nous aurons donc ici deux stratégies : offensive et défensive (Nous reviendrons sur le patron de conception juste après.). Une partie est composée d’un ou deux JeuDecarte.

La classe Score apparait rattachée à Partie avec une flèche en pointillé car c’est une classe que nous rajoutons optionnellement. Son but est de stocker les meilleurs scores, c’est-à-dire le nom du joueur et le nombre de tour qu’il a fallu réaliser pour décrocher la victoire et ceci grâce à la méthode enregistrerScore().

La classe JeuDeCarte comporte un attribut unJeuDeCarte qui est une collection (ArrayList), à l’image d’un jeu de carte rangé par couleur. La méthode battreLeJeu() retourne une collection de jeuDeCarte ou les cartes ont été rangée de manière aléatoire. La méthode distribuerCarte() répartie les cartes dans les différentes zones de jeu. La méthode echangerLesCartes(unIdJoueur :String) permet d’échanger des cartes entre la main et la zone CarteVisible pour un joueur donné. La classe JeuDeCarte est un singleton et composée de 52 cartes. Une carte appartient à un seul JeuDeCarte.

La classe Carte comporte les attributs protégés suivants : libelleCarte, idCarte, couleurCarte, NumPaquetCarte et enfin force de la carte. Dans le jeu de carte, on peut attribuer une puissance à chaque carte. Par exemple la carte 3 est de force 3, la carte 4 de force 4 et le roi de force 13.

La classe CarteSpeciale hérite de Carte et dipose de la méthode actionCarteSpéciale(forceCarte) qui en fonction de la carte (Deux, Sept, Huit, As, Dix) passée en paramêtre va exécuter un bloc d’instruction différent correspondant à l’effet propre de cette carte.

Ensuite, la classe ZoneDeJeu est composé de 1 à 2 JeuDeCarte et comportant un attribut protected nbCarte renseignant le nombre de carte contenue par une zone. Ainsi que l’attribut statique jeuDeCarteMelange de type jeuDeCarte. Les classes Pioche, CarteVisible, CarteCachees, Tapis, Main héritent toutes de ZoneDeJeu. Ces classes ont toutes été redefinies concernant la collection de carte.

La classe Main est relié à la classe Joueur de telle sorte qu’une main corresponde à un et un seul joueur et un joueur n’a qu’une et une seule main. Elle possède la méthode choisirCarte(LeOuLesCartes :Collection de Carte) qui permet de jouer une ou plusieurs cartes de même valeurs.

Le Patron de conception Strategy :

Dans l’optique de séparer les algorithmes de la classe joueur virtuel tout en continuant de jouer normalement les parties, le patron de conception Strategy est donc la solution à adopter.

Il permettra au joueur virtuel de posséder deux types d’attitudes : La première aura tendance à utiliser les cartes spéciales dès qu’il le pourra tandis que l’autre tentera de les conserver pour les utiliser dans des circonstances précises. En clair cela permet d’adapter à une situation donnée, les actions du joueur virtuel elle-même adaptée à cette situation. Nous disposerons donc d’une interface Stratégie comportant une méthode d’exécution comme effectuer() ; les classes qui vont implémenter cette interfaces redéfiniront alors la méthode selon le mode du joueur virtuel choisi. On peut même imaginer un changement de mode au cours d’une partie pour un joueur virtuel.