Université de Caen Normandie

Méthode de Conception :

Rapport : Jeux de cartes, Blackjack

Professeur :
M. Yann Mathet

Groupe
Daouda TRAORÉ 2B
Papa Samba TOURÉ 2A
Erwan Phillipe MENSAH 2B
Amadou BAH 2A

Licence 3 Informatique : 25 novembre 2022

Table des matières

0.1 Introduction : 1
0.2 Principe du BlackJack : 1
0.3 Présentation du projet : 1
0.3.1 application cartesMVC : 1
0.3.2 L'application BlackJack : 3
0.4 Classe Blackjack : 4
0.5 Organisations du M-VC(Modèle - VueControleur) : 5
0.5.1 Le Modele : 5
0.5.2 Vue-Controleur : 5
0.6 Utilisation des design Patterns 6
0.6.1 Le pattern Factory : 6
0.6.2 Le pattern Adapter : 6
0.6.3 Le pattern Observer 7
0.7 Conclusion 7

0.1 Introduction :

En licence 3 informatique à l'université de Caen Normandie, la méthode de conception est une discipline indispensable nous permettant de mieux embrassé la programmation orientée Object.
En outre, cette discipline permet aussi d'apprendre les principes des bonnes pratiques de conception comme la modularité, la maintenabilité ainsi que l'étude et la mise en pratique du design patterns.

Ainsi, il nous est proposé de résoudre le jeu de cartes(blackjack) en utilisant le langage java ainsi que ses techniques citées un peu plus hautes.

C'est dans ce contexte que nous nous sommes réunis par groupe de 4 étudiants pour atteindre cet objectif.

0.2 Principe du BlackJack :

Le blackjack consiste à s'approcher du nombre 21 sans jamais le dépasser tout en gagnant contre la dealer en obtenant une main qui est supérieure à celle du dealer.

Pour cela, chacun des joueurs doit choisir le montant de sa première mise : l'argent qu'il mise en première fois.
Le dealer qui est en général le modérateur du jeu distribue deux cartes à chaque joueur et à lui-même.
Une des cartes du dealer doit être face cachée obligatoirement.
On examine la valeur de chaque carte puis on analyse les mains que contient chaque joueur.
Quand c'est au joueur courant de jouer, il peut tirer une carte tout en n'espérant ne pas dépasser le nombre 21.
Il peut aussi doubler sa mise s'il considère sa main satisfaisante. Une main est satisfaisante si les deux cartes détenues par le joueur sont paires. Il doit donc les séparer en deux
mains différentes avec chacune ayant une mise.
Après le passage de tous les joueurs, le dealer joue. Sa main doit obligatoirement être supérieure à 17 sinon, il tire une nouvelle carte.
Le joueur gagnant doit avoir obligatoirement une main supérieure à celle du dealer et inférieure ou égale à 21.

0.3 Présentation du projet :

Le projet consiste à la réalisation de deux applications.
L'une étant une application de cartes qui servira pour plus tard comme librairie modélisant une carte et un deck de cartes de taille quelconque et proposant une vue sur ces derniers.
L'autre étant le jeu blackjack en lui-même utilisant la librairie de carte réalisée avant.

0.3.1 application cartesMVC :

Cette application dont l'arborescence est représenté ci-dessous est composé essentiellement de deux classes :

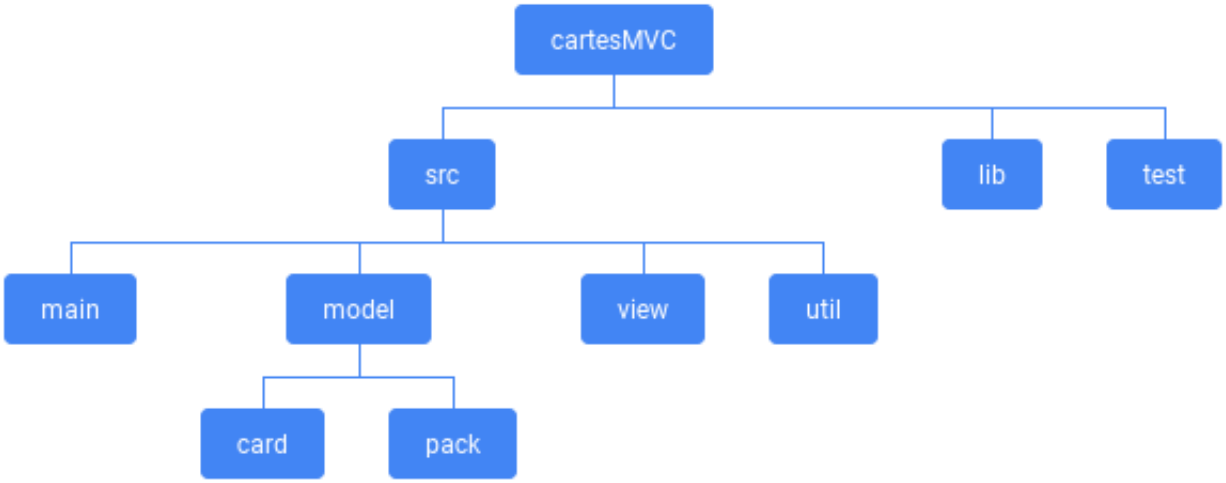

FIGURE 1 – Arborescence du dossier CartesMVC

Représentation d'une carte :

On s'est aidé d'une classe **Card** pour modéliser une carte de jeu qui s'illustre par sa **hauteur**, sa **couleur** et un attribut de type boolean qui nous montre si la carte est face cachée.
La couleur et la hauteur étant des enums permettant ainsi de limiter les possibilités d'instanciation.

Représentation d'un paquet de cartes :

Étant donné qu'un paquet de cartes est un ensemble de cartes, on le représente par une liste de cartes. Dans notre cas, nous avons utilisé une LinkedList car étant meilleure pour les opérations d'insertion et de suppression.
On pourra ainsi grâce à certaines méthodes faire de nombreuses opérations comme : Ajouter ou retirer en début, fin ou à n'importe position aléatoire.
Mélanger les cartes à l'aide de la méthode **shuffle** de la classe Collections, mais aussi scinder le paquet de carte de manière aléatoire.

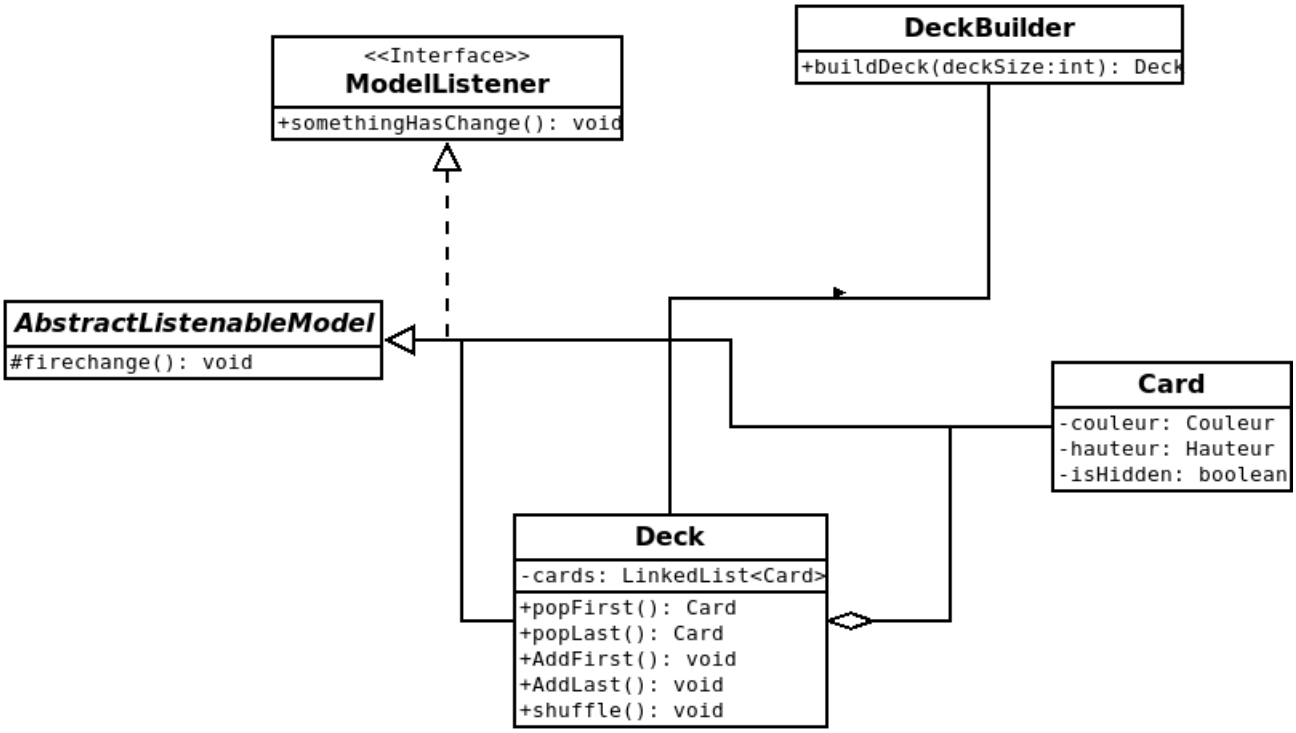

FIGURE 2 – FIGURE-2 : UML de l'application Carte

0.3.2 L'application BlackJack :

L'application BlackJack réprésente le jeu en lui même. Elle est composée de plusieurs packages.

- Le modèle contenant les sous packages **player** et **game**.
- Le package view contenant les classes assurant la vue du jeu

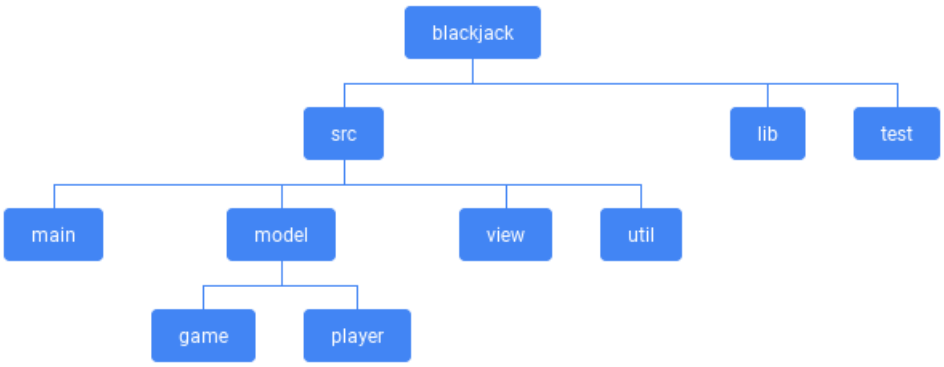

FIGURE 3 – Figure-3 : Arborescence du dossier blackjack :

— La classe Hand :

C'est la main qu'un participant pourra détenir après distribution des différentes cartes.
Cette main est calculée à grâce à la méthode **calculateHand** qui donne la valeur de la main d'un joueur à chaque instant T.

— La classe Player :

Constitue la représentation d'un jouer, un joueur est caractérisé par son **nom**, sa **main**, le **montant de sa mise** et la quantité d'argent qu'il a en réserve.
Les méthodes **hit, stand, push** représentent les différentes actions qu'il peut durant la partie.

— La classe dealer :

La classe représentant le dealer.
Un dealer est un joueur sans mise et sans banque. Mais pour des raisons de cohérence, nous avons préféré faire que la mise du dealer soit toujours égale à celle du joueur et que sa banque soit infinie.
Les deux classes implémentent l'interface **PlayerInterface** et étendent **AbstractPlayer** qui est elle-même écoutable.

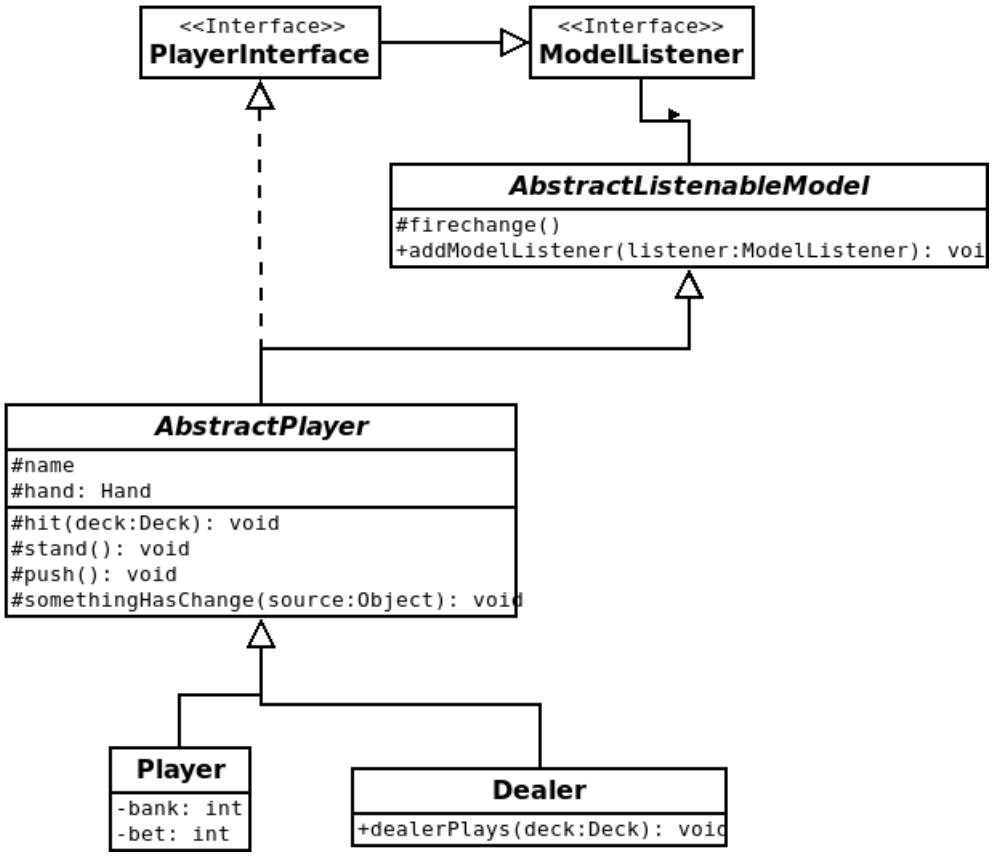

FIGURE 4 – Figure 4 :Package player

0.4 Classe Blackjack :

Classe contenant les méthodes essentielles pour le déroulement d'un jeu de blackjack.
Implémente l'interface **BlackJackInterface** et est écoutable. La classe est composé d'un joueur, du

dealer et d'un deck de cartes.
Le fonctionnement du blackjack est une succession de méthodes allant du début (blackjack.initialise) à la fin (gameOver).
On crée un paquet de cartes grâce à la **factory DeckBuilder**, un Joueur par une instanciation de Player et un Dealer par une instanciation Dealer.
Avec la méthode **shuffle** de la classe Deck, on mélange toutes les cartes puis on les distribue au joueur et au dealer en s'aidant de la méthode **addFrom** de la classe Hand.
La deuxième carte du dealer est cachée au début puis révélée une fois les mises faites. Puis le joueur par les méthodes **hit** ou **stand** décide de sa prochaine action. Le dealer révèle sa carte cachée puis tire une carte tant que sa main ne soit pas supérieure ou égale à 17.

Grâce à la méthode **checkRoundWinner** nous vérifions qui est le gagnant de ce round. On vérifie si le joueur possède un blackjack (une main égale à 21) et non le dealer, alors le joueur a gagné. Si c'est le dealer qui le possède, alors c'est lui qui gagne et le joueur perd sa mise. Si tous les deux possèdent une valeur de main similaire, on annule la mise et on recommence.

0.5 Organisations du M-VC(Modèle - VueControleur) :

0.5.1 Le Modele :

Le modèle contient l'ensemble de la logique de l'application. Il est isolé de la vue et du controlleur

Le modèle du package cartesMVC est constitué des classes Card et Deck.

Le modèle du package blackjack est celui qui est le plus important. Il est composé des classes comme : le joueur, la main du joueur et le Dealer qui constituent le cœur du jeu en lui-même.

0.5.2 Vue-Controleur :

Dans notre application pour des raisons de simplicité nous avons préféré intégré le contrôleur directement dans la vue.
Faisant ainsi une vue-contrôleur et non vue et contrôleur chacun séparer. Le contrôleur est chargé d'agir sur le modèle. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer. La vue fait l'interaction avec l'utilisateur. Sa première tâche est d'afficher les données qu'elle a récupérées auprès du modèle.
Plusieurs deux sont proposées, une sur console avec la classe **BlackjackConsoleView** et une autre graphique. Les classe **BlackjackView** et **ScoreView** représentent le gros de la vue, elle porte une référence sur la classe **Blackjack** et sont abonnées à elle grâce au pattern **Observer** se mettant ainsi à jour à chaque fois le modèle change.
Le contrôleur est l'ensemble des boutons contenu dans la classe **GameGUI**. Chacun de ces boutons lancent une action affectant directement le modèle.

0.6 Utilisation des design Patterns

Un design pattern est un arrangement caractéristique de modules, reconnu comme bonne pratique pour résoudre un problème de conceptiond'un logiciel.
Etant donné que la plupart d'entre eux ont été appris en cours, on a eu dans notre projet à en utiliser quelques-uns :

0.6.1 Le pattern Factory :

Elle permet d'instancier des objets dont le type est dérivé d'un type abstrait. C'est pour celà on a créer une classe **DeckBuilder** qui permet de construire un paquet de cartes sous format LinkedList à l'aide de sa méthode **buildDeck()**. Dans ce cas, à chaque fois on a besoin un paquet de cartes, on appelle la méthode buildeck(deckSize) en lui passant la taille souhaité(32 ou 52). Nous déléguons ainsi l'instanciation de la classe à une classe autre que elle même.

0.6.2 Le pattern Adapter :

Le pattern est un patron de conception (design pattern) de type structure (structural). Il permet de convertir l'interface d'une classe en une autre interface que le client attend(Wikipedia). Nous utilisons ce pattern afin de réaliser une représentation sous la forme d'un tableau de l'état actuelle de la partie. Le tableau affiche le nom des joueurs, la valeur de leurs mains, le montant de leur mise et combien il leur reste en banque. Comme dit plus haut la mise du dealer sera toujours similaire à celle du joueur et sa banque sera toujours infinie. Adaptateur est du même super type que l'adapté c'est-à-dire **BlackjackInterface** et étends la classe abstraite **AbstractTableModel** et possède une référence sur **Blackjack**.

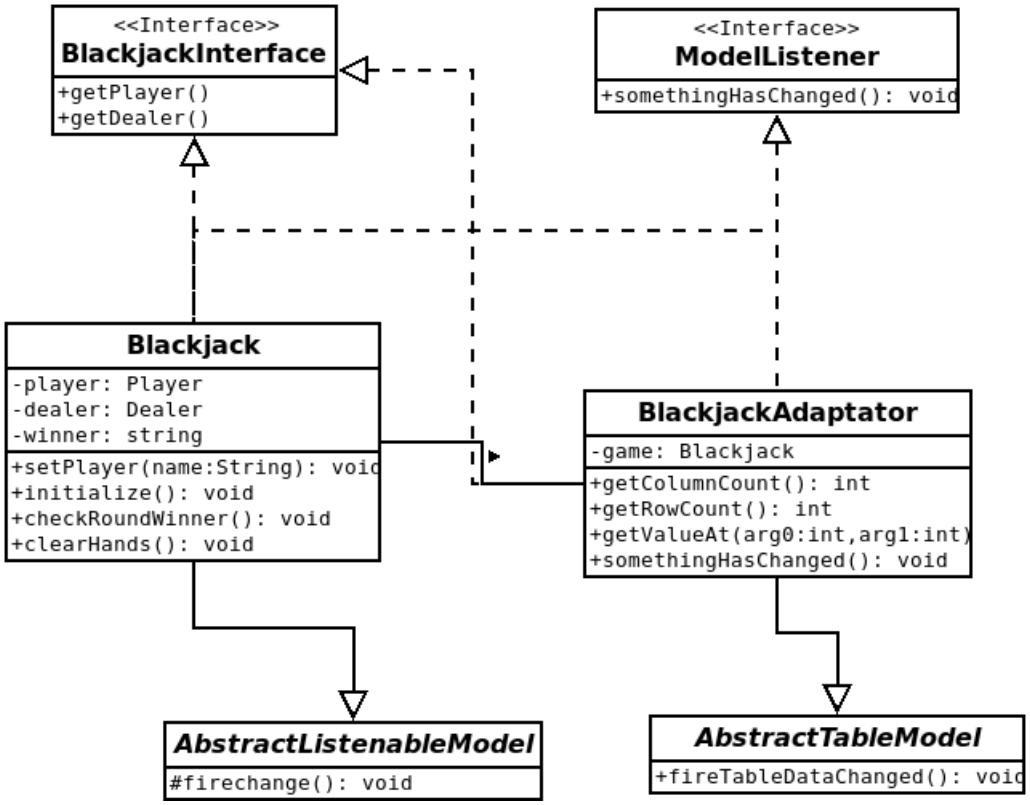

FIGURE 5 – Utilisation du pattern adapter pour représenter l'état du jeu

0.6.3 Le pattern Observer

On a eu également à utiliser le pattern observer ou un objet écoutable possède une liste d'écouteurs et les notifie automatiquement à chaque changement de son état.
L'application étant assez complexe une chaine de écoutables-écouteurs a dû se former. Elle va du plus bas de l'échelle, c'est-à-dire la classe **Card** et se propage jusqu'à la vue **BlackjackView**.

0.7 Conclusion

La réalisation de ce projet nous aura permis d'approfondir et d'un peu plus des concepts vus la première fois en L2 et de commencer à assimiler une approche différente sur la manière de développer des applications qui l'utilisation de design pattern.
Durant ce projet d'autres patterns comme **Decorator, Strategy et Template** auraient pu être implémentés mais nous avions préféré nous limiter à ceux décrit plus haut.