**MVC** :

Le Modèle :

Il y a l’implémentation un système d'écouteurs dans le modèle pour que d'autres entités puissent "écouter" les changements du modèle.

Il y a plusieurs « listeners » pour le modèle avec chacun un événement*.*

*// La vue reçoit un évènement (notification) indiquant que le modèle a changé.*

1.CardListener avec son event qui est CardChangedEvent = quand on va devoir enlever une carte de la SousListe du joueur.

2.PlayerListener avec son event qui est PlayerChangedEvent = quand on change de joueur après un tour.

*//En ayant 2 listeners qui sont des interfaces nous pouvons les mettre dans la vue.*

*Chaque auditeur possède des méthodes différentes qui sont appelées pour traiter leurs événements.*

*// Moi : oral*

Le contrôleur avertit la vue à chaque changement de Sous Liste Joueur et son Id.

Et parallèlement la vue met à jour par le contrôleur les SousListes des joueur par le contrôleur.

Le contrôleur : des vues abstraites

Le contrôleur doit le moins possible être dépendant de Swing : on a donc créé une classe abstraite représentant une vue du plateau (AbstractViewCard). C’est a partir de cette classe qu’on va pouvoir connecter la Vue et le Contrôleur.

Le contrôleur ne manipule que des objets de type View et non plus de type Swing.

La vue sera représentée par une JFrame (JFrameViewCard).

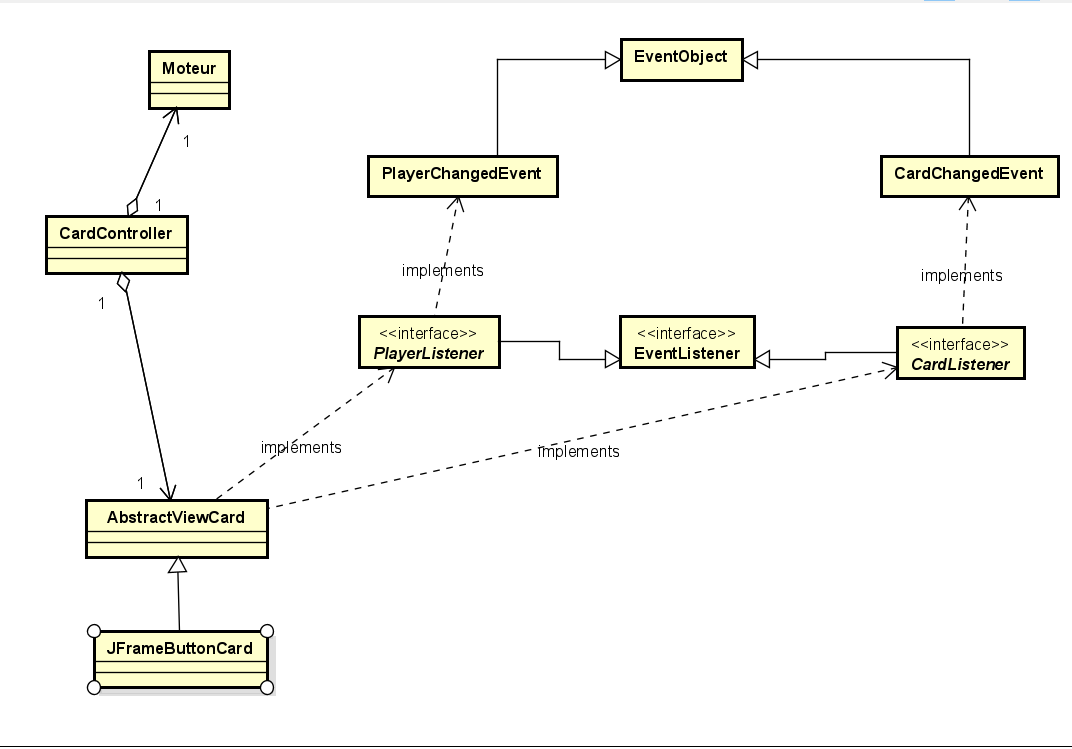
On a mis le moteur et le contrôleur en Thread. Comme ça, on peut attendre que le moteur s’initialise pour que le contrôleur appelle la vue. Et par la suite attendre que la vue finisse un tour complet (=1 Age) pour recommencer et « jouer ». C’est sur le point de jeu que nous avons eu du mal.

Dans le run du contrôleur, on demande au contrôleur d'afficher la vue.

Lors de la création de la vue, celle-ci s'enregistre en tant que listener du modèle.

Classe lanceur la classe "main" de l'application:

Elle crée un nouveau modèle, crée un nouveau contrôleur en lui passant le modèle. Et « start » les 2 threads qui sont le modèle et le contrôleur.

Récapitulatif MVC utilisé :