

La Main de Poker #1

22 septembre 2021



Objectifs du TD

Après avoir identifié les premières slices permettant de progresser sur le projet, nous allons implémenter les premières slices durant ce TD.

[Voir le sujet du projet](#)

Proposition de slices

Slice 1 : la carte gagnante !

- Le programme permet de saisir deux mains composées d'une carte chacune puis de les comparer.
- Une carte comporte un numéro allant de 1 à 13. Une carte n'a pas de couleur.
- La main gagnante est affichée à la fin de l'exécution.
- Pas de gestion de l'égalité.

Slice 2 : l'égalité !

- Si les deux mains comportent une carte de la même valeur, le programme doit afficher "Egalite".

Slice 3 : et de deux !

- Il est désormais nécessaire de saisir deux cartes pour chaque main.
 - Le programme doit toujours être en mesure d'identifier la carte la plus haute.
-



Implémentation de la première slice

Un programme en quatre phases

Que nous implémentions une ou dix slices, voire le programme dans son ensemble, nous allons toujours suivre l'enchaînement de phase qui suit:

1. Lecture, interprétation et validation de la saisie utilisateur.
2. Identification des combinaisons de cartes (paire, suite, carte la plus haute, etc...) de chaque main et comparaison des mains.
3. Affichage des résultats.

Chaque phase du programme évoluera et se complexifiera au fur et à mesure de l'implémentation des slices.

Classes principales ?

La carte

Créez une classe Card contenant une valeur contenue dans un entier.

La main

Créez une classe Hand contenant une carte de type Card ainsi qu'un nom de type String.

La partie

Créez une classe Game contenant deux mains de type Hand.



Corps du programme

Créez une classe `PokerHand` contenant une méthode (fonction) `main` (`public static void main(String [] args) {}`).

Cette méthode sera systématiquement exécutée au démarrage du programme.

Afin de rendre la lecture du code simple, cette méthode doit être la plus légère possible et doit se contenter d'orchestrer les différentes phases du programme (les trois phases décrites plus haut). Le détail de l'implémentation de ces phases sera délégué à d'autres classes.

Lecture, interprétation et validation de la saisie utilisateur

Dans une classe dédiée (`InputReader`), créez une méthode publique `readGame()` permettant de lire les données saisies sur l'entrée standard.


Vous pouvez lire l'entrée standard en utilisant une instance de la classe [Scanner](#). La méthode `nextLine()` permet de récupérer les entrées une ligne à la fois.

Construisez une instance de `Game` à partir de deux instances de `Hand` (elles-mêmes construites à partir des données récupérées). Ajoutez des constructeurs à vos classes pour pouvoir les instancier. N'oubliez pas de donner un nom unique à votre main ("1" ou "2" par exemple).

Lors de la construction de vos instances de classe, vous pouvez à tout moment arrêter le programme en cas de données invalide. Pour ce faire, utilisez [System.exit\(1\)](#).

La méthode `readGame()` retourne un objet de type `Game`.

Éditez la classe `PokerHand` pour que la méthode `readGame()` soit appelée. Vous stockerez la partie retournée dans une variable dédiée.



Identification et comparaison de main

Dans la classe `Game`, créez une méthode publique `compete()` permettant de comparer les mains de poker.

Comparez les valeurs de la carte de chaque main et retournez la main ayant la plus haute carte.

Éditez la classe `PokerHand` pour que la méthode `compete()` de la partie créée plus tôt soit appelée. Stockez la main gagnante dans une variable dédiée.

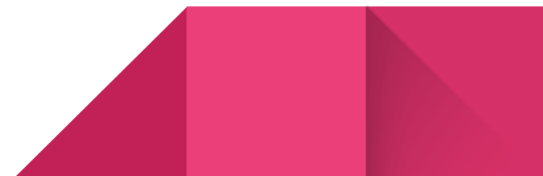
Affichage des résultats

Dans une classe dédiée (`ResultPrinter`), créez une méthode publique `showResult()` permettant d'afficher le résultat de la partie. Cette méthode prend une main en paramètre.

Affichez le nom de la main victorieuse grâce au code suivant:

```
System.out.println("<Message de victoire>");
```

Éditez la classe `PokerHand` pour que la méthode `showResult()` soit appelée avec en paramètre la main victorieuse.



Implémentation de la seconde slice

L'objectif de cette slice est de permettre le support de la détection des égalités ainsi que leur affichage.

Transporter l'information d'égalité

Actuellement la méthode *compete()* de la classe *Game* retourne la main victorieuse. En cas d'égalité, il n'y a pas de main victorieuse. Pour symboliser cette absence de vainqueur nous pouvons changer le type de retour de la méthode : en utilisant le type *Optional<Hand>*, nous indiquerons qu'il est possible qu'il n'y ait pas de vainqueur.

Suite à ce changement votre code ne devrait plus compiler.


Faite les actions suivantes pour que cela recompile à nouveau:

- Éditez le code de *compete()* afin qu'il retourne un optional contenant la main gagnante à la place de juste la main.
- Changez le type de la variable contenant la main victorieuse dans la méthode *main()* de *PokerHand*.
- Changez le type du paramètre de la méthode *showResult()* de la classe *ResultPrinter* afin qu'il accepte un *Optional<Hand>* au lieu d'un *Hand*. Le code de *showResult()* doit être modifié : la méthode [get\(\)](#) sur l'objet *Optional* vous permettra de récupérer la main.

Détection de l'égalité

Maintenant que nous savons comment faire transiter l'information d'égalité, il ne nous reste plus qu'à la détecter !

Faites évoluer la méthode *compete()* de la classe *Game* afin que l'égalité soit détectable. En cas d'égalité retournez un optional vide (*empty*).



Afficher l'égalité

Faites évoluer la méthode *showResult()* de la classe *ResultPrinter*. Si l'optionel passé en paramètre est vide cela signifie qu'il y a égalité. Affichez le bon texte dans la console dans ce cas là.



Implémentation de la troisième slice

Pour cette troisième slice, nous allons implémenter le support d'une seconde carte dans chaque main. Cela va impliquer plusieurs changements.

Classes principales ?

Actuellement la classe `Hand` ne contient qu'une seule carte. Nous voulons désormais pouvoir en contenir plusieurs.

Remplacez la variable contenant l'unique carte par une variable contenant une [liste](#) de cartes.

Le code ne devrait plus compiler. Faites le nécessaire pour qu'il re-compile de nouveau.

Lecture et interprétation de deux cartes

Nous devons rendre possible pour un utilisateur de saisir plusieurs cartes.

Lors de la construction de vos mains à partir de la saisie utilisateur, découpez la chaîne de caractère en deux morceaux ([split\(\)](#) est votre ami).

Itérer sur chaque sous-chaînes pour créer les deux cartes de la main.

Détecter la meilleure carte

Actuellement la méthode `compete()` de `Game` ne regarde qu'une seule carte de chaque main pour les comparer. Ajoutez une méthode `highestCard()` à la classe `main` qui retourne la carte la plus haute.

Utilisez la carte retournée par `highestCard()` pour effectuer la comparaison des mains.

