Programmation orientée objet
TP n°2

Objectif

L'objectif de cette partie du TP est de réaliser un programme plus complexe comportant plusieurs classes. Le but est de jouer au jeu de black-jack.

Nous allons dans cette partie nous intéresser à la distribution et au mélange d'un paquet de 52 cartes.

1. Écriture de la classe Carte

La classe carte comportera deux attributs privés :

Une valeur de type numérique et une couleur de type numérique. Pour les couleurs et les valeurs supérieures à 10, vous utiliserez les constantes de classe suivantes (à placer au début de votre classe)

```
public final static int PIQUES = 0,       // Codes pour les 4 couleurs
                        COEURS = 1,
                        CARREAUX = 2,
                        TREFLES = 3;

    public final static int AS = 1,      // codes pour les cartes non numériques
                        VALET = 11,     //   les valeurs 2 à  10 ont leur
propre valeur
                        DAME = 12,
                        ROI = 13;
```

- Écrire un constructeur `public Carte(int valeur, int couleur)` qui crée une carte de valeur passée en paramètres ;
- Écrire les accesseurs `public int getCouleur()` et `public int getValeur()` qui retournent les attributs spécifiés ;

- Écrire les méthodes `public String getCouleurCommeChaine()` et `public String getValeurCommeChaine()` qui retournent respectivement la couleur et la valeur de la carte comme chaîne. On retournera ?? si la valeur est indéterminée ;
- Écrire la méthode public String toString() qui retourne la valeur et la couleur de la carte comme cela est décrit ci-dessous.

Testez votre classe avec le programme suivant, expliquez la ligne Carte c2 = new Carte(Carte.ROI,Carte.TREFLES).

A quoi correspondent Carte.ROI et Carte.TREFLES ?

```
public class Main {

    public static void main(String [] args){
    Carte c1 = new Carte(10,2);
    Carte c2 = new Carte(Carte.ROI,Carte.TREFLES);


    System.out.println(c1);
    System.out.println(c2.getCouleur());
    System.out.println(c2);
}
}
```

Expliquez l'affichage

10 de Carreaux

3

Roi de Trèfles

1. Écriture de la classe MainJoueur

Nous allons maintenant écrire la classe MainJoueur qui permet de conserver jusque 6 cartes. La classe MainJoueur contiendra 2 attributs, le premier attribut sera un tableau **privé** de Carte et un attribut privé entier nbCartes qui servira à garder la trace du nombre de cartes dans la main du joueur.

```
public class MainJoueur {

    private  Carte [] mainJ;
    private int nbCartes;
}
```

- Écrire la constructeur de la classe MainCarte qui initialisera à 0 le nombre de cartes du joueur et qui créera le tableau mainJ (`mainJ = new Carte[6];`)
- Écrire la méthode public void clear() qui « vide » la main du joueur. Le nombre de cartes sera remis à 0 et chaque valeur du tableau aura la valeur **null** ;
- Écrire la méthode public void ajouterCarte(Carte c) qui ajoute la carte c à la main du joueur si c'est possible ;
- Écrire la méthode public void enleverCarte(int pos) qui supprime la carte à la position passée en paramètre (mise à la valeur null dans le tableau). Attention si on enlève la carte 2, il faut déplacer les cartes qui étaient « situés » au-dessus, la carte 3 devient la carte 2, la carte 4 devient la carte 3 et ainsi de suite (*cf* exemple ci-dessous) ;
- Écrire la méthode public Carte getCarte(int position) qui retourne la carte située à la position passée en paramètre ;
- Écrire la méthode public void trieParValeur() qui trie la main du joueur par valeur (cf exemple ci-dessous) ;
- Écrire la méthode public String toString() qui permet d'afficher la main du joueur.
- Tester votre programme avec le programme principal ci-dessous :

```
public class Main {

    public static void main(String [] args){
    Carte c1 = new Carte(10,2);
    Carte c2 = new Carte(Carte.ROI,Carte.TREFLES);

    Carte c3 = new Carte(8,Carte.TREFLES);
    Carte c4 = new Carte(6, Carte.TREFLES);

    MainJoueur mj = new MainJoueur();

    mj.ajouterCarte(c1);
    mj.ajouterCarte(c2);
    mj.ajouterCarte(c3);
    mj.ajouterCarte(c4);

    System.out.println(mj);

    mj.enleverCarte(2);

    System.out.println(mj);

    }

}
```

qui doit vous donner l'affichage suivant :

```
$ java Main
Il y a 4 cartes
10 de Carreaux
Roi de Trèfles
```

```
8 de Trèfles
6 de Trèfles

Il y a 3 cartes
10 de Carreaux
Roi de Trèfles
6 de Trèfles
```

Écriture de la classe Deck

1. **La classe Deck comportera deux attributs privés :**

Un attribut que l'on appellera **deck** qui correspondra à un tableau de 52 cartes et un attribut privé de type entier **cartesUtilisees** qui correspondra au nombre de cartes « utilisées » dans ce deck.

```
private Carte [] deck;   // Un tableau de 52 cartes correspondant à un jeu complet
    private int cartesUtilisees; // Combien de cartes ont été tirées de ce deck ?
```

- Écrire un constructeur `public Deck()` qui crée l'intégralité des 52 cartes du jeu dans l'ordre. L'attribut cartesUtilisees sera bien entendu positionné à la valeur 0 ;
- Écrire la méthode `public void shuffle()` qui permet de mélanger la totalité du jeu. Pour ce mélange, on procédera de la manière suivante. On parcourt les cartes une à une et pour chaque carte, on l'échange avec une autre carte du jeu choisie aléatoirement. Vous pouvez utiliser la classe Random pour réaliser le tirage d'un nombre aléatoire. Il faudra rajouter dans ce cas, une instance de Random dans les attributs ;
- Écrire la méthode `public Carte distribuerCarte()` qui retourne une carte et incrémente l'attribut cartesUtilisees ;
- Écrire la méthode `public int cartesRestantes()` qui retourne le nombre de cartes disponibles dans le deck ;
- Enfin écrire la méthode `public String toString()` qui retourne la totalité des cartes présentes dans le deck ;
- Vous testerez votre classe avec le programme principal suivant :

```
public class Main {

    public static void main(String [] args){

    Deck deck = new Deck();
    deck.shuffle();

    System.out.println(deck);
```

```
    }

}
```

Vous devriez obtenir un affichage proche de celui ci-dessous :

```
Liste de cartes
9 de Cœurs
As de Cœurs
3 de Cœurs
Dame de Piques
8 de Piques
2 de Cœurs
Valet de Piques
8 de Carreaux
5 de Trèfles
3 de Carreaux
8 de Cœurs
9 de Carreaux
Valet de Trèfles
2 de Piques
Valet de Cœurs
10 de Carreaux
4 de Cœurs
8 de Trèfles
2 de Carreaux
10 de Cœurs
9 de Trèfles
6 de Piques
4 de Carreaux
4 de Trèfles
As de Carreaux
5 de Piques
Roi de Trèfles
7 de Trèfles
3 de Piques
3 de Trèfles
As de Trèfles
Dame de Trèfles
6 de Carreaux
6 de Cœurs
4 de Piques
10 de Piques
5 de Cœurs
Valet de Carreaux
Dame de Cœurs
Roi de Piques
10 de Trèfles
Roi de Carreaux
Dame de Carreaux
As de Piques
2 de Trèfles
6 de Trèfles
9 de Piques
7 de Cœurs
7 de Piques
5 de Carreaux
7 de Carreaux
Roi de Cœurs
```

Complétion de la classe MainJoueur

Nous allons maintenant compléter la classe MainJoueur de manière à calculer la

nombre de points dans la main du joueur.

Commençons par la règle simplifiée du black-jack (d'après Wikipedia) :

La partie oppose tous les joueurs contre la banque. Le but est de battre le croupier sans dépasser 21. Dès qu'un joueur fait plus que 21, on dit qu'il « saute » ou qu'il « crève » et il perd sa mise initiale. La valeur des cartes est établie comme suit :

- De 2 à 9 → valeur nominale de la carte
- Chaque figure + le 10 surnommées "bûche" → 10 points
- L'As → 1 ou 11 (au choix)

De manière à ce que ce soit un peu plus simple, nous partirons du principe que l'on ajoute 11 si on ne dépasse pas le plafond pour l'as et un si l'ajout de 11 points dépasse le plafond.

Nous rajoutons à notre classe la méthode suivante :

```
public int getBlackjackValeur()
```

qui retourne la valeur de la main de black-jack à partir des règles expliquées ci-dessus.

- Tester votre programme avec le programme principal ci-dessous :

```
public class Main {

    public static void main(String [] args){

    Deck deck = new Deck();
    deck.shuffle();

    MainJoueur mj = new MainJoueur();

    Carte c = deck.distribuerCarte();
    mj.ajouterCarte(c);
    System.out.print(mj);
    System.out.println(mj.getBlackJackValeur() + "\n");

    c = deck.distribuerCarte();
    mj.ajouterCarte(c);
    System.out.print(mj);
    System.out.println(mj.getBlackJackValeur() + "\n");


    c = deck.distribuerCarte();
    mj.ajouterCarte(c);
    System.out.print(mj);
    System.out.println(mj.getBlackJackValeur() + "\n");
```

```
    }
}
```

qui doit vous donner l'affichage suivant :

```
$ java Main
Il y a 1 cartes
9 de Piques
9

Il y a 2 cartes
9 de Piques
9 de Carreaux
18

Il y a 3 cartes
9 de Piques
9 de Carreaux
5 de Carreaux
23
```

Écriture du programme de jeu simplifié

Le but du jeu est d'obtenir une valeur la plus proche possible de 21 sans jamais dépasser cette valeur. Le programme aura besoin d'un objet de la classe Deck et de deux objets de type MainJoueur (un pour le joueur et un pour le croupier).

1. Deux cartes sont distribuées au croupier et deux cartes sont ensuite distribuées au joueur. Si le croupier a une valeur de 21 en main il gagne. Si le joueur a une valeur de 21, c'est lui qui gagne (c'est ce qu'on appelle le black-jack). Notez qu'en cas d'égalité, c'est le croupier qui gagne ;
2. Si le jeu n'est pas terminé, le joueur peut prendre des cartes dans sa main. Il voit bien évidemment ses cartes et une des deux cartes du croupier. Le joueur peut choisir de tirer une carte et de l'ajouter à sa main ou de s'arrêter ;
3. Si le joueur dépasse 21, il perd automatiquement ;
4. Lorsque le joueur s'arrête, le croupier peut tirer des cartes. La règle est simple, tant que sa main est inférieure ou égale à 16, le croupier tire des cartes. Le joueur à ce niveau voit les cartes du croupier. Le gagnant est déterminé : si le croupier a dépassé 21, il a perdu et le joueur gagne. Sinon, si le croupier a une main de valeur supérieure ou égale au joueur, il gagne, autrement le joueur a gagné.

À vous de programmer tout ça !