# Td : Les collections

## Exercice 1

On veut gérer des informations sur des joueurs de tennis, les entraîneurs et les tournois. Pour cela, on dispose des informations suivantes:

- ✓ Les joueurs et les entraîneurs sont des personnes avec un numéro, un nom et un prénom.
- ✓ Un entraîneur dispose en plus d'un indice sur son coefficient victoire/défaite qui est mis à jour régulièrement.
- ✓ Un joueur est caractérisé par un classement ATP.

Les joueurs participent à des tournois. Ces derniers sont définis par un numéro, la ville où se déroule le tournoi, une dotation (en dirhams), une date de début et de fin puis d'autres champs possibles. On donne un numéro, un nom à chaque ville et on doit également pouvoir obtenir le nom du pays de cette ville.
Vous devez implémenter le programme en Java et fournir un jeu d'essai dans la méthode main qui va créer des joueurs, des entraîneurs avec les tournois et villes.
Contraintes : créer 4 joueurs, 2 entraîneurs, 2 tournois dans 2 villes différents.
Votre système doit permettre de poser les questions suivantes :

- ➢ qui entraîne qui,
- ➢ numéro du tournoi joué actuellement pour un joueur donné,
- ➢ classement ATP d'un joueur.

Les structures de données permettant de stocker les informations sur les joueurs, tournois, etc. doivent correspondre à des objets du type ArrayList ou autres

Remarque : il faut construire les classes, les méthodes ainsi que d'ajouter les champs qui doivent être logiquement présents dans chaque classe.

## Exercice 2

Écrivez une fonction qui construit une collection *triée* contenant *n* nombres entiers (représentés par des objets Integer) tirés au hasard dont la valeur est comprise entre 0 et 100. Ensuite, la fonction doit afficher la collection construite afin qu'on puisse constater qu'elle est bien triée.

**a.** Dans la première version de la fonction, la collection est de type LinkedList.

**public void arrayListTrie(String n)**

b. Dans la deuxième version, la collection est de type TreeSet.

**public void treeSetTrie(String n)**

**Attention** : Il faut gérer les exceptions possibles.

## Problème

L'objectif du problème est de définir des classes pour permettre de représenter des jeux de cartes.
Le jeu de cartes traditionnel comprend les cartes suivantes :

| | |
|---|---|
| Piques : | As • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9 • 10 • Valet • Dame • Roi |
| Cœurs : | As • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9 • 10 • Valet • Dame • Roi |
| Carreaux : | As • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9 • 10 • Valet • Dame • Roi |
| Trèfles : | As • 2 • 3 • 4 • 5 • 6 • 7 • 8 • 9 • 10 • Valet • Dame • Roi |
| Autres : | 2 jokers |
| Couleurs: | pique ♠, cœur ♥, carreau ♦, trèfle ♣ |

Il existe 13 cartes de chaque couleur dans un jeu de 54 cartes (incluant deux jokers) ce qui fait que si l'on considère que le valet à un rang de onze, la dame un rang de douze, le roi un rang de treize.

## Partie 1

La classe Carte, représentant une carte à jouer traditionnelle, est définie comme suit :

```
public class Carte implements Comparable<Carte> {

        // ….VARIABLES DE CLASSE….
static public final String[ ] NOMS_COULEURS = { "Pique", "Coeur", "Carreau", "Trèfle" };
static public final String[ ] CARS_COULEURS = { "♠", "♥", "♦", "♣" };
static public final String[ ] NOMS_RANGS = { "Joker", "As", "2", "3", "4", "5", "6", "7",
                                              "8","9", "10", "Valet", "Dame", "Roi"};
        // ….VARIABLES D'INSTANCE….
private  String couleur; // Couleur de la carte
private final int rang; // Rang de la carte
private final int force; // Force de la carte.
        // ….PARTIE METHODES....
}
```

- Les tableaux NOMS_COULEURS, CARS_COULEURS (les caractères associés aux couleurs des cartes) et NOMS_RANGS serviront à former des chaînes de caractères destinées à l'affichage de cartes.
- Le rang représente un ordre naturel invariable pour les cartes.

| Carte | Joker | AS | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Valet | Dame | Roi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RANG | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

- La force représente la valeur relative d'une carte par rapport aux autres. Une carte l'emporte sur une autre carte si et seulement si elle est plus forte. Les forces doivent être définies en fonction des jeux. Elles permettent de définir un ordre particulier pour un jeu (par exemple, l'As peut avoir une force supérieure au Roi dans certains jeux). Par défaut, la force est égale au rang.

### Question

Compléter la classe Carte en implémentant les méthodes d'instance qui sont données ci-dessous tout en gérant les exceptions possibles:

| | |
|---|---|
| 1- **public Carte(String couleur, int rang)** | Construit une carte en donnant sa couleur et son rang. La force de la carte sera égale à son rang. Tout rang extérieur à [0,13] sera ramené à 0 (Joker).<br>@param **couleur** String : La couleur de la carte<br>@param **rang** int : Le rang de la carte |
| 2- **public Carte(String couleur, int rang, int force)** | Construit une carte en donnant sa couleur, son rang et sa force.<br>Tout rang extérieur à [0,13] sera ramené à 0 (Joker).<br>@param **couleur** String : La couleur de la carte<br>param **rang** int : Le rang de la carte<br>@param **force** int : La force de la carte |
| 3- **public String getCarCouleur()** | Fournit le caractère associé à la couleur de cette carte.<br>@return String : Le caractère associé à la couleur. |
| 4- **public String getNomRang()** | Fournit le nom associé au rang pour la carte courante.<br>@return String: Le nom du rang |
| 5- **public String toString( )** | Fournit la représentation complète associée à cette carte, sous la forme :<br>"*10 de Trèfle :: 10*♣" **ou** "*Dame de Coeur :: Dame*♥").<br>@return String. |
| 6- **public boolean equals(Carte autreCarte)** | Permet de comparer l'égalité de deux cartes, qui s'exprime par l'égalité des couleurs et des rangs.<br>@param **autreCarte** Carte.<br>@return boolean |
| 7- **public int compareTo(Carte autreCarte)** | Permet de comparer deux cartes<br>**1-** La carte courante et la deuxième carte sont égales s'ils ont la même couleur et le même rang.<br>**2-** Si non la carte courante est supérieure à une autre carte si la couleur de la carte courante et de la deuxième carte sont égales et le rang de la carte courante est supérieur au rang de la deuxième carte.<br>**3-** Si non la carte courante est inférieure à la deuxième carte.<br>@param **autreCarte** Carte<br>@return int « 0 ; 1 ; -1 » |
| 8- Ajouter les méthodes : **les setteurs** et **les getteurs** qui sont associées aux champs de la classe | |

## Partie 2

Pour jouer aux cartes, on a besoin d'une classe qui gère un jeu de cartes.
La classe JeuCartes sera définie come suit :

```
public class JeuCartes {

        // VARIABLES D'INSTANCE
private ArrayList<Carte> jeu; // le jeu de cartes

        // ...PARTIE METHODES...
}
```

**Questions** :
Compléter la classe JeuCartes en implémentant les méthodes d'instance suivantes tout en gérant les exceptions possibles:

| | |
|---|---|
| 9- **private JeuCartes( )** | Construit un jeu de cartes vide. |
| 10- **private void addAllCartes(int rangDebut, int rangFin, String couleur_X)** | Ajoute toutes les cartes du rang rangDebut au rang rangFin et de couleur couleur_X.<br>La force des autres cartes est initialisée par défaut.<br>@param **rangDebut** int : Rang de la première carte<br>@param **rangFin** int : Rang de la dernière carte<br>@param **couleur** String : couleur des cartes |
| 11- **private void addCarte(Carte carte)** | Ajoute cette carte au jeu si elle n'existe pas.<br>@param **carte** Carte : La carte à ajouter |
| 12- **public String toString ( )** | Retourne une représentation textuelle complète du jeu de cartes sous la forme :<br>*Jeu de 52 cartes : [As de Pique :: 1♠, Deux de Pique :: 2♠, ..., Roi de Trèfle :: Roi♣]*<br>@return String : La chaîne formatée représentant le jeu de cartes. |
| 13- **static public JeuCartes createJeuCartes32(boolean forceAS_MAX)** | Construit et retourne un jeu de 32 cartes en indiquant si la force de l'As doit être supérieure à toutes les autres cartes (soit une force de 14),sinon sa force est la plus petite (soit 1).<br>Les forces des autres cartes sont initialisées par défaut. Le premier rang des autres cartes qui sont différant de l'As est égal à 7.<br>@param boolean : **forceAS_MAX** true si la force de l'As est la plus grande, false sinon<br>@return JeuCartes : Le jeu de cartes |
| 14- **static public JeuCartes createJeuCartes52( boolean forceAS_MAX)** | Construit et retourne un jeu de 52 cartes en indiquant si la force de l'As doit être supérieure à toutes les autres cartes (soit une force de 14), sinon sa force est la plus petite (soit 1).<br>Les forces des autres cartes sont initialisées par défaut.<br>@param **forceAS_MAX** true si la force de l'As est la plus grande, false sinon<br>@return Le jeu de cartes |
| 15- **public void melangeJeu( )** | Mélange le jeu de cartes aléatoirement. |

| | |
|---|---|
| 16- **public void insertCardEnDessous(Carte card)** | Insère une carte donnée en dessous du jeu (c.à.d: la première position).<br>@param **card** Carte: La carte à insérer |
| 17- **public Carte getUneCardDuDessus( boolean doitEtreSupprimer)** | Retourne la carte du dessus du jeu (la dernière carte) en la supprimant du jeu si c'est demandé (lecture ou extraction).<br>S'il n'y a pas de carte, cette méthode retourne null.<br>@param **doitEtreSupprimer** boolean: Faut-il extraire la carte jeu ?<br>@return Carte: La carte du dessus du jeu ou null si le jeu est vide*/ |
| 18- **public ArrayList<Carte> getCardsDuDessus( int nbCartes, boolean doitEtreSupprimer)** | Construit et retourne un tableau de cartes du dessus du jeu.<br>Le tableau de cartes retourné contient exactement le nombre de cartes demandées. Il est complété par des éléments nuls si le jeu n'a pas assez de cartes.<br>L'ordre des cartes du tableau retourné est tel que la première carte correspond à la dernière du jeu (i.e. ordre inverse de position des cartes dans le jeu).<br>@param **nbCartes** int: Nombre de cartes à récupérer<br>@param **doitEtreSupprimer** boolean: Faut-il extraire les cartes du jeu ?<br>@return ArrayList<Carte>: Les cartes du dessus du jeu en ordre inverse, complétées par des éléments nuls si le jeu n'a pas assez de cartes. |