Java td/tp n°9
Interface

Problème
Le but de l'exercice est de créer un programme Java simulant un jeu de cartes.
Soit l'interface *DonneesCarte* décrivant des constantes caractéristiques d'une carte à jouer :

```
public interface DonneesCarte
{
        public static final int NB_VALEURS    = 8 ;

        public static final int   [] VALEURS      = {7, 8, 9, 10, 11, 12, 13, 14} ;
        public static final String[] LIBELLES   = {"sept ", "huit ", "neuf ", "dix  ", "valet",
                                    "dame ", "roi  ", "as   "} ;
        public static final int NB_COULEURS  = 4 ;
        public static final String[] COULEURS  = {"trèfle", "carreau", "cœur", "pique"}  ;
}
// fin interface DonneesCartes
```

Soit la classe *Carte* décrivant une carte à jouer et rendant les services suivants (voir l'interface *Comparable* en annexe) :

public class *Carte* **implements** *DonneesCarte, Comparable*

Constructor Summary

Carte() Contructeur vide
Carte(Carte c) Constructeur par copie
Carte(int uneValeur, String uneCouleur) Initialise la Carte courante

Method Summary

int	**compareTo**(Object o) retourne -1, 0 ou 1 selon que la *Carte* courante est inférieure, égale, ou supérieure a la carte référencée par *o*
boolean	**equals**(Object o) retourne *true* si la Carte référencée par *o* est égale a la *Carte* courante
String	**getCouleur**() retourne la couleur de la *Carte* courante
int	**getValeur**() retourne la valeur de la *Carte* courante
void	**init**() initialise la *Carte* courante
String	**toString**() retourne la chaîne de caractères représentant la *Carte* courante

1. Sachant que la classe *Carte* dispose de la méthode privée suivante :
   private int indiceCouleur()
   // retourne l'indice de la couleur de la Carte courante dans le tableau COULEURS
   // de l'interface *DonneesCartes*

   écrire la méthode *compareTo* sachant qu'une *Carte c1* est supérieure à une *Carte c2* si la valeur de *c1* est supérieure à la valeur de *c2*. En cas d'égalité des valeurs *c1* est inférieure à *c2* si la couleur de *c1* précède la couleur de *c2* dans le tableau COULEURS de l'interface *DonneesCarte*.

2. Soit l'interface *PaqueCartes* décrivant les services que doit rendre un paquet de *Cartes* :

public interface **PaquetCartes :**

Method Summary	
boolean	**estVide**() retourne *true* si le *Paquet* est vide, *false* sinon
void	**insererCarteDessous**(Carte c) insère la carte en dessous (1ère position) du *Paquet*
void	**insererCarteDessus**(Carte c) insère la carte au dessus (dernière position) du *Paquet*
void	**melangerCartes**() mélange le *Paquet* de cartes
int	**nbCartes**() retourne le nombre de cartes dans le *Paquet*
Carte	**regarderCarteDessous**() *throws Exeption* retourne (sans la retirer) la carte située au dessous du *Paquet (*une *Exception* est créée et déléguée si le paquet est vide)
Carte	**regarderCarteDessus**()*throws Exeption* retourne (sans la retirer) la carte située au dessus du *Paquet (*une *Exception* est créée et déléguée si le paquet est vide)
Carte	**retirerCarteDessous**()*throws Exeption* retire et retourne la carte située au dessous du *Paquet (*une *Exception* est créée et déléguée si le paquet est vide)
Carte	**retirerCarteDessus**()*throws Exeption* retire et retourne la carte située au dessus du *Paquet (*une *Exception* est créée et déléguée si le paquet est vide)

i) Ecrire la classe *PaquetCartesListe* implémentant l'interface *PaquetCartes* :

a. Déclarer une variable d'instance cartes de type *ArrayList* (cf annexe)
b. Ecrire le constructeur vide de façon à ce que la variable *cartes* désigne un paquet de cartes initialement vide.
c. Ecrire les méthodes *estVide* et *nbCartes*,
d. Ecrire les autres méthodes
e. Écrire la méthode *toString*
f. Écrire une méthode *figures()* qui supprime du paquet courant et retourne la liste des cartes représentant des figures (valet, dame, roi, as).

(il n'est pas demandé d'écrire la méthode *mélangerCartes*).

ii) Ecrire la classe *PaquetCartesTableau* implémentant l'interface *PaquetCartes* :

a. Déclarer une variable d'instance *Carte[] cartes*, et une variable d'instance *int pos* qui désigne la position dans le tableau cartes de la prochaine carte à ajouter (donc *pos-1* désigne la position de la dernière carte dans la tableau).
b. Écrire le constructeur vide de façon à ce que la variable *cartes* désigne un paquet de cartes contenant au maximum 52 cartes, de plus *pos* est initialisé à *0*).
c. Écrire les méthodes *estVide* et *nbCartes*,
d. Écrire les autres méthodes
e. Écrire la méthode to *String()*

Exercice complémentaire

On souhaite jouer à la *Bataille* régit par les règles suivantes :

le jeu se joue à 2 joueurs. On dispose d'un paquet de 32 cartes mélangées au hasard, 16 cartes sont distribuées à chaque joueur.

A chaque tour de jeu les joueurs retirent une carte du dessus de leur paquet et les comparent. Le joueur ayant retiré la plus forte carte remet les 2 cartes en dessous de son paquet. En cas d'égalité chaque joueur reprend sa carte et la place en dessous de son paquet. Le jeu est terminé lorsqu'un joueur n'a plus de cartes (il est perdant).

Ecrire une classe Bataille implémentant l'interface *DonneesCartes*. La classe *Bataille* contiendra 2 variables d'instances *mainA* et *mainB* désignant les paquets de cartes des 2 joueurs.

g. Ecrire le constructeur vide de telle façon que *mainA* et *mainB* désignent 2 paquets de cartes vides.
h. Ecrire les méthodes suivantes :

private public static PaquetCartes *creerJeuCartes ( )*
// créé et retourne un jeu de 32 cartes (utiliser les constantes de l'interface DonneesCartes)
private void distribuerCartes*(PaquetCartes unJeu)*
// retire les cartes du paquet unJeu pour les mettre tour à tour dans mainA et mainB.
public void jouer*( )*
// créer un paquet de 32 cartes, le distribue à mainA et mainB, puis joue à la bataille
// selon les régles énoncées ci-dessus.

TP

Récupérer les fichiers *.java* dans bouthinon/apprentissage/apprentissage2/tp9 et écrire ou compléter les classes *Carte* (méthode *compareTo*), *PaquetCartesListe*, *PaquetCartesTableau* et *Bataille*. Tester les classes au fur et à mesure qu'elles sont complétées.

ANNEXE

Interface Comparable

Method Summary	
int	**compareTo**(Object o) retourne respectivement *–1, 0* ou *1* si l'objet courant est inférieur, égal ou supérieur à l'objet référencée par o.

public class **ArrayList** extends AbstractList
implements List, RandomAccess, Cloneable, Serializable

Constructor Summary
ArrayList() Construit une liste vide

Method Summary	
void	**add**(int index, Object o) insert *o* dans la liste au rang spécifié par *index*
boolean	**add**(Object o) ajoute *o* à la fin de la liste
Object	**get**(int index) retourne l'élément de la liste situé au rang *index*
boolean	**isEmpty**() retourne *true* si la liste est vide
Object	**remove**(int index) retire de la liste l'élément situé au rang *index*
int	**size**() retourne le nombre d'éléments dans la liste
Methods inherited from class java.util.AbstractList	
public ListIterator **listIterator**() // retourne un itérateur sur la liste courante	

Note : tout itérateur permet de parcourir une liste et dispose de méthodes permettant d'ajouter et supprimer "proprement" des éléments de/dans la liste pendant le parcours. Tout itérateur implémente l'interface suivante :

Interface ListIterator :

Method Summary	
void	**add**(Object o) insère o en fin de liste
boolean	**hasNext**() retourne true s'il y a un élément derrière le curseur géré par l'itérateur.
Object	**next**() retourne l'élément situé derrière le curseur de l'itérateur et avance le curseur d'un cran
void	**remove**() retire de la liste l'élément lu par le dernier *next( )*