Université Côte d'Azur
PeiP2

mercredi 29 septembre 2021
Durée: 3h

Algorithmique et Programmation Java
Travaux Dirigés – Séance n. 4

Dans le TD précédent, vous avez défini la classe `Carte` pour représenter une carte à jouer. On souhaite maintenant représenter l'ensemble d'un jeu de 52 cartes. Les cartes seront mémorisées dans un tableau.

Rappels : Un tableau est un type structuré qui permet de regrouper des éléments de même type accessibles par un *indice*. Par exemple, la déclaration :

```
1 int []t;
```

déclare un tableau `t`. Toutefois, les éléments du tableau ne sont pas créés. `t` est simplement une référence. Pour créer les éléments, par exemple 10 entiers, on écrira :

```
1 t = new int[10];
```

Les indices des élements sont compris entre 0 et `t.length-1`. Le nombre total d'éléments est donné par `t.length`. Ainsi, `t[0]` correspond au premier élément, et `t[t.length-1]` au dernier. Notez que, `t[-4]` ou `t[20]` provoque une erreur puisque les indices sont faux.

exercice 1) Dans le fichier `Jeu52.java`, écrivez la classe `Jeu52` pourvue de la variable privée `jeu` de type tableau de `Carte`. On la munira d'une constante de classe qui indique le nombre de cartes (ici 52) du jeu.

exercice 2) Dans cette classe, écrivez le constructeur `Jeu52()` qui initialise le tableau avec les 52 cartes. On mettra les cartes dans l'ordre "croissant" de l'`enum Couleur`, puis pour chaque couleur, dans l'ordre de l'`enum Valeur` définies au TP précédent.

exercice 3) Ajoutez à cette classe la méthode `toString()` qui renvoie une `String` formée de la représentation textuelle des 52 cartes à jouer.

exercice 4) Dans la méthode `main` d'une classe nommée `Jeu` (ou `TestJeu`), déclarez et initialisez la variable `lesCartes` avec un objet de type `Jeu52`.

exercice 5) Affichez sur la sortie standard le jeu de 52 cartes.

exercice 6) Ajoutez dans `Jeu52` un accesseur public permettant de récupérer la référence de n'importe quelle carte du jeu. Elle a la signature suivante et on s'assurera que l'indice est dans les bornes du tableau de cartes, si tel n'était pas le cas, la méthode renverrait null :

```
1 public Carte getCarte(int indice);
```

On souhaite mélanger les cartes. Pour cela, on va échanger 52 fois, deux cartes tirées au hasard dans le tableau.

exercice 7) Ajoutez à la classe `Jeu52`, la méthode `mélanger` selon l'algorithme précédent. Utilisez la classe `java.util.Random` qui permet de créer des générateurs aléatoires de valeurs. Consultez la documentation de cette classe.

exercice 8) Dans la classe `Jeu`, mélangez le paquet de cartes et affichez-le sur la sortie standard.

On va vouloir ordonner le paquet de cartes.

exercice 9) Cela suppose que deux cartes sont comparables, afin de décider laquelle des deux est plus petite que l'autre. Retravaillez si besoin la méthode `compareTo` de la classe `Carte` afin de

prendre en compte l'ordre (arbitraire) de couleur, et pas seulement l'ordre de valeur d'une carte afin de la comparer à celle référéncée en paramètre. Exemple le valet de trèfle doit être considéré comme plus petit que le valet de carreau, et plus petit que toute dame indépendamment de la couleur de la dame, bien qu'on ait mis pour la dame la valeur de 10, comme pour un valet. On notera que c'est parce qu'on a énuméré les couleurs et les valeurs dans un ordre que nous considérons "croissant", que nous pouvons présumer que telle couleur ou telle valeur est plus petite qu'une autre.

Notez qu'ayant défini une méthode int compareTo(Carte c) dans la classe Carte, toute carte a la caractéristique de pouvoir être comparée à une autre carte. De ce fait, il se trouve que cette méthode `compareTo` qu'on a déjà codée est celle qu'on doit implémenter si l'on prétend qu'une classe implements l'interface dont le nom est `java.lang.Comparable`. On peut donc ajouter à notre définition de la classe Carte cette information de typage. Ne cherchez pas les raisons du type indiqué entre < et >, c'est un peu complexe et hors sujet pour ce semestre.

exercice 10) Modifiez en conséquence l'entête de la classe Carte :

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

exercice 11) Ajoutez à la classe `Jeu52`, la méthode `ordonner` qui ordonne le paquet de cartes selon l'ordre initial. C'est-à-dire, qui trie le tableau par couleur croissante, et dans chaque couleur par la valeur associée à chaque carte. Pour cela, l'algorithme de tri proposé est le plus simple : c'est le tri dit "par sélection". A chaque tour, pour la place courante $i (0 <= i < taille)$ il cherche parmi les valeurs non encore triées, celle qui est la plus petite, et l'insère à la place courante i (en échangeant deux cartes, celle qui était à la place i et celle qui doit prendre la place i). Vous ajouterez dans la classe une méthode privée qui permettra de réaliser l'échange de deux cartes du tableau.

Vous allez maintenant utiliser la planche à dessin pour visualiser toutes les cartes du jeu.

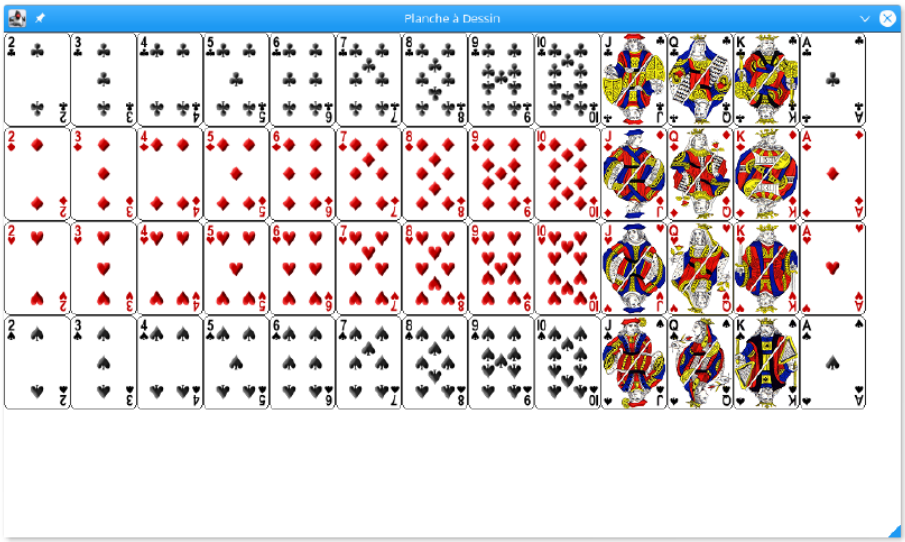
Afin que les cartes ne se superposent pas, il faut connaitre la largeur et hauteur d'une carte. Comme elles sont toutes de même dimension, c'est inutile d'exécuter un code similaire dans chaque instance de Carte, dont le but est de récupérer le nombre de pixels en largeur ou hauteur de l'Image associée à la carte : autant ne le faire qu'une seule fois, et donc, au moment où la classe Carte est chargée par la JVM.

exercice 12) Ajoutez à la classe Carte une constante de classe (cad, un attribut final static) indiquant la largeur en pixel d'une carte, en se servant d'un des fichiers d'image récupérés pour le TD3. Par exemple, en se servant de l'image qui correspond à un dos de carte (fichier "dos.gif"), on peut instancier l'objet Image et ensuite récupérer la largeur. De même pour connaitre la hauteur. Pour initialiser cet attribut, nous allons utiliser un bloc de code `static`. Posez vous la question : aurait il été possible d'utiliser un autre fichier d'images, par exemple, un fichier dont le nom serait fonction de valeur et couleur ? Si cette question est trop complexe, on peut aisément utiliser un attribut largeur, hauteur pour chaque instance de Carte, qui soit positionné à une valeur en pixel réaliste.

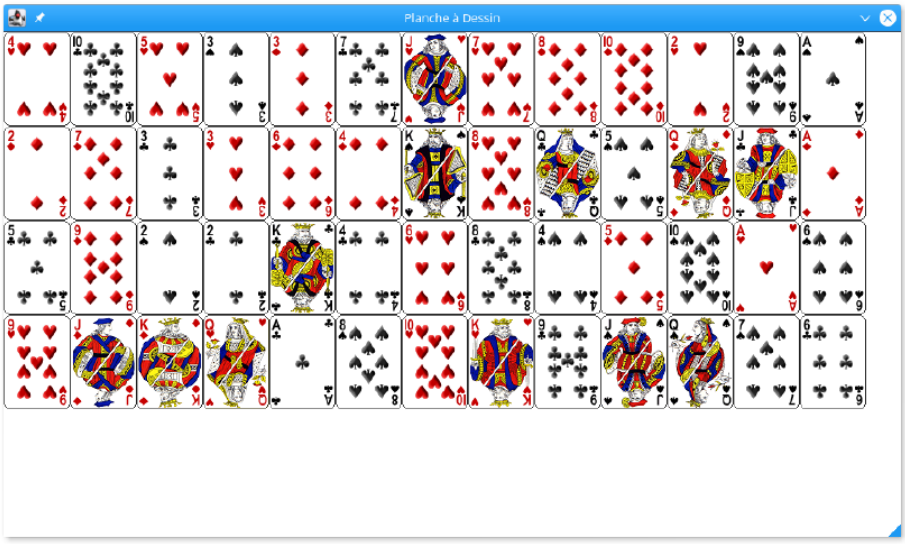
exercice 13) Dans la méthode `main`, créez un objet `PlancheADessin` et visualisez toutes les cartes en prenant soin que les cartes ne se superposent pas. Pour cela, codez une méthode dans `Jeu52` ayant la signature

```
1 public void dessiner(PlancheADessin pl)
```

. Vous devez obtenir quelque chose comme :



exercice 14) Mélangez le jeu de cartes et visualisez à nouveau toutes les cartes (vous pouvez vous aider d'un appel à `nextInt()` sur un objet de la classe `java.util.Scanner`, si vous voulez avoir le "temps" de voir chaque état du jeu). Vous devez obtenir quelque chose comme :



Triez le jeu, et relancez l'affichage afin de vérifier visuellement que votre tri fonctionne bien.

On veut maintenant représenter des joueurs de cartes.

exercice 15) Dans un fichier `Joueur.java`, déclarez la classe `Joueur` avec deux variables privées, `nom` et `mesCartes`. La variable `nom` de type `String` est le nom du joueur, alors que la variable `mesCartes` de type tableau de `Carte` contiendra les cartes du joueur.

exercice 16) Écrivez le constructeur de cette classe qui initialise la variable `nom`, mais, qui n'initialise PAS ENCORE la variable `mesCartes`.

exercice 17) Écrivez la méthode `prendreMesCartes` qui met dans le tableau `mesCartes` N cartes prises dans le jeu de cartes. Elles seront prises d'un seul coup dans le jeu de cartes. Il faudra donc

instancier le tableau `mesCartes` de la taille voulue dans cette méthode. L'en-tête de cette méthode sera donc le suivant :

```
1 public void prendreMesCartes(Jeu52 jeu, int de, int à);
```

exercice 18) Dans la méthode main, testez prendreMesCartes. On supposera qu'un joueur prendra un nombre de cartes égal au nombre total divisé par le nombre total de joueurs (ce qui, pour un jeu de 52 cartes et 4 joueurs fera que chaque joueur prendra 13 cartes consécutives).

exercice 19) Écrivez la méthode `montrerCartes` qui visualise dans une planche à dessin, à une hauteur h donnée en paramètre, le nom du joueur et ses cartes. L'en-tête de cette méthode est le suivant :

```
1 public void montrerCartes(PlancheADessin pad, double h)
```

C'est en pilotant la valeur passée pour le paramètre h que, depuis le main, vous assurerez que les cartes de chaque joueur ne se chevauchent pas avec celles du joueur précédent ou suivant sur la planche à dessin.

---

exercice 20) Dans la méthode `main` de la classe `Jeu` déclarez un tableau de 4 joueurs.

exercice 21) Mélangez le jeu de cartes, et faites en sorte de distribuer toutes les cartes en fonction du nombre de joueurs, ici $52/4 = 13$ cartes à chacun des 4 joueurs.

exercice 22) Visualisez les noms des joueurs et leurs cartes dans la planche à dessin. Vous devez obtenir quelque chose comme :

On veut pouvoir « retourner » une carte, c'est-à-dire si la face est visible, on affiche sur la planche à dessin le dos de la carte, et inversement, si c'est le dos de carte qui est visible, on affichera à la place sa face. Parmi les images de cartes que vous avez récupérées pour le TD3, le fichier `dos.gif` correspond à l'image d'un dos de cartes. Il faudra par contre associer à chaque Carte une seconde image (de la classe `Image`) (donc un second attribut), correspondant à la photo d'un dos de carte, en plus de celle correspondant à la face de la carte. On ne peut, en effet, partager le même objet Image montrant le dos, entre toutes les cartes, car chaque Image a ses propres coordonnées de placement (x,y) sur la planche à dessin (même si elle affiche le même fichier .gif).

Pour la suite, il vous faudra ajouter un attribut à Carte afin de savoir si c'est actuellement le dos ou la face qui est visible sur la planche à dessin. Cet attribut devrait être privé, et donc, ajoutez un accesseur public si par la suite et depuis une autre classe que `Carte`, il vous est utile de connaitre si la face ou le dos d'une carte est actuellement affiché.

exercice 23) Complétez la classe `Carte` avec la méthode `retournerCarte` qui retourne la carte courante dans la planche à dessin passée en paramètre. Cette méthode à l'en-tête suivant :

```
1 public void retournerCarte(PlancheADessin pad)
```

Vous pourrez utiliser la possibilité qu'une image déjà positionnée en (x,y) pourra être remplacée par une autre à la même position (x,y). Pour cela, il faudra être capable d'avoir accès aux coordonnées (x,y) où la carte est actuellement dessinée. On utilisera également la méthode de suppression d'un objet `Dessinable` de la planche à dessin. Il suffit ensuite d'invoquer la méthode `dessiner` de Carte (faite au TD3) à nouveau, afin que la carte réapparaisse et au même (x,y) sur la planche à dessin.

exercice 24) Testez depuis main votre méthode `retournerCarte` sur quelques cartes du jeu.

Cette dernière partie du TD4 ne sera pas traitée en séance, et un délai supplémentaire pour la rendre sera octroyé. Elle constitue un mini projet.

On souhaite implémenter une version très simplifiée du blackjack : `https://fr.wikipedia.org/wiki/Blackjack\_(jeu)`

On se souvient que chaque carte a une valeur associée (entre 2 et 20) (cf l'enum `Valeur`). Le but de cette version simplifiée du blackjack est qu'un (seul) joueur retourne un certain nombre de cartes au hasard parmi les N cartes qu'il possède, et qu'il s'arrête lorsque le total des cartes retournées atteint au minimum la valeur de 15 ou qu'il n'ait plus envie de jouer.

Toutefois, on demandera à chaque fois à l'utilisateur de confirmer le fait de retourner chacune des cartes à l'aide d'une boîte de dialogue produite par le code suivant que vous insérerez dans une méthode privée de la classe de Test (donc, en dehors de `main`), en lui donnant la signature suivante : `private static int retourner()` et dont le code sera

```
1 int ret = JOptionPane.showConfirmDialog
2             (null,
3          "Retourner 1 carte au hasard ?",
4          "Dialogue",
5          JOptionPane.OK_CANCEL_OPTION);
6 return ret;
```

utilisant `javax.swing.JOptionPane`.

Si l'utilisateur clique sur le bouton OK, la variable `ret` sera égale à `JOptionPane.OK_OPTION`, sinon elle sera égale à `JOptionPane.CANCEL_OPTION`.

exercice 25) Selon la valeur de `ret`, si elle est égale à `JOptionPane.OK_OPTION`, retournez 1 carte au hasard parmi celles non encore retournées. Pour cela, vous implémenterez dans la classe Joueur une nouvelle méthode nommée

```
1 public Carte choisirCarteCacheeAuHasard()
```

Elle devra s'assurer qu'une carte ne peut être choise que si son dos est actuellement affiché. Cette méthode sera donc utilisée dans le main, pour simuler le fait qu'un des joueurs s'est mis à jouer au blackjack.

D'autres petites méthodes pourraient être utiles ! Par exemple, avant que le joueur ne commence la partie de Blackjack, il faudrait que toutes ses cartes soient mises face cachée.

Bien sûr, il faudrait afficher le score cumulé pour ce joueur. On pourra simplement l'afficher sur la sortie standard. Ou mieux, l'afficher à côté du nom du joueur sur la planche à dessin.

Vous pourrez obtenir quelque-chose comme :

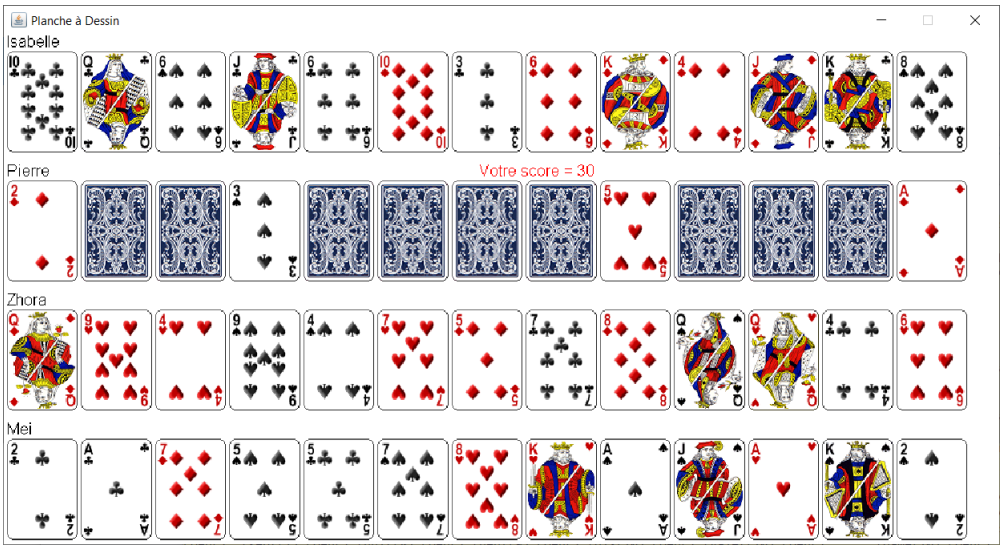


Planche à Dessin
Isabelle
Pierre
Votre score = 30
Zhora
Mei