Laboratoire 7

DURÉE PRÉVUE : 8H00

OBJECTIFS

Au terme de ce laboratoire, l'étudiant sera capable de :

→ déclarer un tableau à une dimension

→ manipuler un tableau à une dimension quelque soit le type de ses éléments (*int*, *String*, ...)

→ transmettre un tableau à une dimension à une fonction

AVANT-PROPOS

Pour réaliser les exercices de ce laboratoire, il vous sera demandé d'utiliser des nombres aléatoires à plusieurs reprises. Vous aurez donc besoin des fonctions de la classe **Aleatoire** :

1. Ajoutez un package nommé **util** dans le package **src** du projet **prb**
2. Importez la classe **Aleatoire** dans ce package

EXERCICE 1 : MANIPULATIONS DE TABLEAUX

Cette première série d'exercices à réaliser sur le site web CodingBat va vous permettre de vous familiariser aux manipulations de base sur les tableaux à une dimension.

PREMIER EXERCICE

Rendez-vous sur la page *http://codingbat.com/java/Array-1*.

Cliquez ensuite sur le premier exercice nommé « **firstLast6** ».

Voici une traduction de l'énoncé : *« Étant donné un tableau d'entiers, retournez vrai si le nombre 6 apparaît en première ou en dernière position du tableau. Le tableau contient au moins un élément »*.

Le code à compléter est :

```
public boolean firstLast6(int[] nums) {

}
```

Cliquez sur le bouton « Go » pour tester votre code. Répétez l'opération jusqu'à ce que tous les tests soient au vert !

CONSIGNES

1. Réalisez les 5 exercices suivants
2. **sameFirstLast** : *« Étant donné un tableau d'entiers, retournez vrai si le tableau contient au moins un élément et si le premier élément est égal au dernier »*
3. **commonEnd** : *« Étant donnés deux tableaux d'entiers, a et b, retournez vrai s'ils ont le même élément en première position ou en dernière position. Les deux tableaux contiennent au moins un élément »*
4. **sum3** : *« Étant donné un tableau de trois entiers, retournez la somme des éléments du tableau (Amélioration possible : calculez la somme à l'aide d'une boucle for...each) »*
5. **rotateLeft3** : *« Étant donné un tableau de trois entiers, retournez un tableau équivalent avec les éléments décalé d'une position vers la gauche, par exemple le résultat est [2, 3, 1] pour le tableau [1, 2, 3] »*
6. **reverse3** : *« Étant donné un tableau de trois entiers, retournez un tableau équivalent avec les éléments en ordre inverse, par exemple le résultat est [3, 2, 1] pour le tableau [1, 2, 3] »*

EXERCICE 2 : MANIPULATIONS DE TABLEAUX À L'AIDE DE BOUCLES

Cette deuxième série d'exercices à réaliser sur le site web CodingBat va vous permettre de vous familiariser aux traitements qui peuvent être réalisés sur les tableaux à une dimension à l'aide de boucles.

PREMIER EXERCICE

Rendez-vous sur la page *http://codingbat.com/java/Warmup-2*.

Cliquez ensuite sur le premier exercice nommé « **arrayCount9** ».

Voici une traduction de l'énoncé : *« Étant donné un tableau d'entiers, retournez le nombre d'apparitions du nombre 9 dans le tableau »*.

Le code à compléter est :

```
public int arrayCount9(int[] nums) {

}
```

Cliquez sur le bouton « Go » pour tester votre code. Répétez l'opération jusqu'à ce que tous les tests soient au vert !

IMPORTANT

Pour chaque exercice de cette série, CodingBat propose une solution accessible via la bouton *« Show solution »*. Dans votre propre intérêt, essayez de réaliser l'exercice par vous-même avant de regarder la solution.

CONSIGNES

1. Réalisez les 3 exercices suivants
2. **arrayFront9** : *« Étant donné un tableau d'entiers, retournez vrai si l'un des quatre premiers éléments du tableau est 9. Le tableau contient au moins un élément »*
3. **array123** : *« Étant donné un tableau d'entiers, retournez vrai si la séquence des nombres 1, 2, 3 apparaît dans le tableau »*
4. **array667** : *« Étant donné un tableau d'entiers, retournez le nombre de fois que le nombre 6 est suivi soit du nombre 6, soit du nombre 7 dans le tableau, par exemple le résultat est 2 pour le tableau [3, 6, 6, 7] »*

EXERCICE 3 : BLACKJACK

Réalisez un programme qui permet à un joueur de faire une partie de blackjack seul face à un croupier.

[Remarque] Les règles énoncées ci-après sont une simplification des règles existantes !

Le blackjack utilise un jeu de 52 cartes. Chaque carte équivaut à un certains nombre de points :

→ les cartes 2 à 9 valent un nombre de points égal à la valeur de la carte

→ le 10 et les figures (Valet, Dame et Roi) valent 10 points

→ l'As vaut 11 points

L'objectif du blackjack est d'atteindre un score aussi proche que possible de 21 en tirant successivement des cartes. Si le joueur dépasse 21 points, il perd aussitôt. Le joueur peut décider de ne plus tirer de carte supplémentaire s'il se considère suffisamment proche de 21 points. Dans ce cas, celui-ci gagne à condition d'avoir un nombre de points supérieur à celui du croupier.

DÉROULEMENT D'UNE PARTIE

1. Le croupier mélange le jeu de cartes
2. Le croupier donne la 1ère carte du paquet au joueur, dépose la 2e carte face cachée sur la table, donne la 3e carte au joueur et dépose la 4e carte face visible sur la table (les cartes déposées constituent les cartes du croupier)
3. Le joueur et le croupier vérifient s'ils ont 21 points (*blackjack*). Si c'est le cas pour l'un des deux, la partie se termine et cette personne gagne. S'ils ont tous les deux 21 points, la partie se termine sur une égalité. Si aucun des deux n'a 21 points, la partie continue
4. Le joueur peut tirer autant de cartes qu'il le souhaite pour se rapprocher des 21 points. S'il dépasse les 21 points, le joueur perd et la partie se termine. S'il atteint exactement 21 points, son tour s'arrête automatiquement
5. Le croupier tire des cartes jusqu'à ce qu'il atteigne au moins 17 points. Se faisant, si le croupier dépasse les 21 points, le joueur gagne et la partie se termine. Dans le cas contraire, la partie est remportée par celui qui a le plus de points ou débouche sur une égalité

PRÉPARATION

1. Dans le package **util**, ajoutez une classe nommée **TableauChaines**
2. Dans la classe **TableauChaines**, déclarez :

   → une fonction nommée **permuter** qui permet de permuter deux éléments d'un tableau de chaînes de caractères situés aux positions i et j :

```
public static void permuter(String[] tableau, int i, int j)
```

→ une fonction nommée **melanger** qui permet de mélanger les éléments d'un tableau de chaînes de caractères :

```
public static void melanger(String[] tableau)
```

Pour ce faire, parcourez chaque position du tableau en permutant l'élément de la position courante avec un autre élément choisi aléatoirement dans le tableau

Utilisez les fonctions *Aleatoire.aleatoire* et *permuter*

→ une fonction nommée **toString** qui permet d'obtenir une chaîne de caractères représentant les premiers éléments d'un tableau de chaînes de caractères :

```
public static String toString(String[] tableau, int nbElements)
```

Le paramètre `nbElements` indique le nombre d'éléments du tableau à intégrer dans la chaîne

Les éléments doivent être séparés par une virgule. Par exemple :

```
"4 de coeur, 8 de carreau, Roi de pique"
```

3. Ajoutez un package nommé **labo7** dans le package **src** du projet **prb**
4. Dans le package **labo7**, ajoutez une classe nommée **Blackjack** comportant une fonction **main**
5. Dans la classe **Blackjack**, déclarez :

→ la fonction suivante qui permet de créer un jeu de cartes :

```
private static String[] getJeuDeCartes() {
   // Constantes
   final String[] COULEURS = { "coeur", "carreau", "trèfle", "pique" };
   final String[] CARTES = { "2", "3", "4", "5", "6", "7", "8", "9",
                             "10", "Valet", "Dame", "Roi", "As" };

   // Variable pour le résultat
   String[] jeuDeCartes;

   // Créer un jeu de cartes
   jeuDeCartes = new String[COULEURS.length * CARTES.length];
   for (int i = 0; i < COULEURS.length; i++) {
      for (int j = 0; j < CARTES.length; j++) {
         jeuDeCartes[CARTES.length * i + j] = CARTES[j] + " de " +
                                              COULEURS[i];
      }
   }

   // Retourner le jeu de cartes
   return jeuDeCartes;
}
```

→ une fonction nommée **getValeurCarte** qui permet d'obtenir la valeur d'une carte :

```
public static int getValeurCarte(String carte)
```

Utilisez la méthode *substring* de la classe *String* afin d'extraire le début de la chaîne jusqu'au premier espace. Par exemple : `"8 de carreau"` → `"8"` → `8`

Pensez également à utiliser la méthode *equals* de la classe *String*, ainsi que la fonction *Pattern.matches*

→ une fonction nommée **getValeurCartes** qui permet d'obtenir la valeur d'un ensemble de cartes :

```
public static int getValeurCartes(String[] cartes, int nbCartes)
```

Le paramètre `nbCartes` indique le nombre d'éléments du tableau qui doivent être pris en compte dans le calcul des points

Utilisez la fonction *getValeurCarte*

6. Dans la fonction **main**, réalisez le jeu du blackjack tel qu'il a été décrit en début d'énoncé (voir aussi les exemples d'exécution ci-dessous). Pour ce faire, utilisez les fonctions précédemment déclarées dans les classes **TableauChaines** et **Blackjack**

EXEMPLE D'EXÉCUTION 1

Le joueur gagne d'entrée de jeu car il obtient directement 21 points.

```
Vous avez 21 points :
As de trèfle, Dame de coeur

Le croupier a 14 points :
5 de coeur, 9 de pique

Vous gagnez !
```

EXEMPLE D'EXÉCUTION 2

Le joueur perd en souhaitant se rapprocher des 21 points.

[Remarque] Personne n'ayant obtenu 21 points avec les deux premières cartes distribuées, seule la deuxième carte du croupier est visible au joueur

```
Vous avez 12 points :
8 de trèfle, 4 de trèfle

Le croupier a 4 points :
4 de carreau

Tirer une carte : (O)ui/(N)on ? o

Vous avez 16 points :
8 de trèfle, 4 de trèfle, 4 de coeur

Tirer une carte : (O)ui/(N)on ? o

Vous avez 26 points :
8 de trèfle, 4 de trèfle, 4 de coeur, 10 de coeur

Vous perdez !
```

EXEMPLE D'EXÉCUTION 3

Le joueur gagne car le croupier dépasse les 21 points en souhaitant atteindre 17 points.

[Remarque] Les cartes du croupier sont révélées une fois qu'il a terminé de tirer des cartes

```
Vous avez 12 points :
Roi de pique, 2 de trèfle

Le croupier a 6 points :
6 de pique

Tirer une carte : (O)ui/(N)on ? o

Vous avez 19 points :
Roi de pique, 2 de trèfle, 7 de coeur

Tirer une carte : (O)ui/(N)on ? n

Le croupier a 23 points :
9 de coeur, 6 de pique, 8 de pique

Vous gagnez !
```

EXEMPLE D'EXÉCUTION 4

Le joueur perd car le croupier obtient davantage de points.

```
Vous avez 16 points :
5 de coeur, As de trèfle

Le croupier a 10 points :
10 de trèfle

Tirer une carte : (O)ui/(N)on ? o

Vous avez 19 points :
5 de coeur, As de trèfle, 3 de pique

Tirer une carte : (O)ui/(N)on ? n

Le croupier a 20 points :
10 de carreau, 10 de trèfle

Vous perdez !
```

EXERCICE 4 : LE TIRAGE DU LOTTO

La Loterie Nationale souhaite un programme permettant de remplir une grille de Lotto et de consulter son gain après chaque tirage.

En Belgique, le tirage du Lotto a lieu chaque mercredi et samedi. 6 numéros et un numéro bonus sont tirés parmi une série de numéros compris entre 1 et 45. Il faut obtenir au moins 2 numéros gagnants et le numéro bonus pour remporter un gain.

Par souci de simplicité, nous considérerons les gains suivants :

Numéros gagnants	Gain	Rang
6	500.000,00 €	1
5+	75.000,00 €	2
5	1.500,00 €	3
4+	250,00 €	4
4	30,00 €	5
3+	10,00 €	6
3	5,00 €	7
2+	3,00 €	8

Le symbole + représente le numéro bonus

Les combinaisons gagnantes sont ordonnées par rangs. Le rang le plus élevé est le rang 1 et le rang le moins élevé est le rang 8.

LOGIQUE DU TIRAGE

Les numéros tirés doivent être uniques, autrement dit aucun numéro ne peut se répéter.

Si le tirage est réalisé à l'aide des instructions suivantes, l'unicité des numéros n'est pas garantie :

```
int[] tirage = new int[7];
for (int i = 0; i < 7; i++) {
  tirage[i] = Aleatoire.aleatoire(1, 45);
}
```

Pour empêcher la présence de deux numéros identiques dans un tirage, une première solution consiste à recommencer le tirage d'un numéro si le numéro tiré est déjà présent dans le tableau. Cette solution comporte l'inconvénient de reposer sur la "chance" pour tirer un numéro qui n'a pas déjà été tiré.

La méthode expliquée ci-après permet d'éviter ce problème en excluant de la série les numéros précédemment tirés :

1. Placer tous les numéros de la série dans un tableau et stocker la position du dernier élément dans une variable `j`

1	2	3	4	5	...	42	43	44	45

2. Choisir aléatoirement une position p comprise inclusivement entre `0` et `j`

3. Copier le numéro situé à la position p dans le tableau `tirage`
4. Permuter les numéros situés aux positions p et `j`
5. Diminuer `j` de 1
6. Répéter les étapes 2 à 5 tant que le tirage n'est pas terminé

Par exemple, si p = 1 lors de la première itération, le tableau devient :

1	45	3	4	5	...	42	43	44	2

Ensuite, si p = 41 à l'itération suivante :

1	45	3	4	5	...	44	43	42	2

Puis, si p = 4 :

1	45	3	4	43	...	44	5	42	2

Etc.

En diminuant `j` de 1 à chaque fin d'itération, le numéro placé en `j` suite à la permutation est ainsi exclu de la partie du tableau dans laquelle le numéro suivant sera choisi.

PRÉPARATION

1. Dans le package **util**, ajoutez une classe nommée **TableauEntiers**
2. Dans la classe **TableauEntiers**, déclarez :

   → une fonction nommée **contient** qui permet de déterminer si un nombre est présent dans un tableau d'entiers :

   ```
   public static boolean contient(int[] tableau, int a)
   ```

   → une fonction nommée **permuter** qui permet de permuter deux éléments d'un tableau d'entiers situés aux positions i et j :

   ```
   public static void permuter(int[] tableau, int i, int j)
   ```

   → une fonction nommée **toString** qui permet d'obtenir une chaîne de caractères représentant tous les éléments d'un tableau d'entiers :

   ```
   public static String toString(int[] tableau)
   ```

   Les éléments doivent être séparés par une virgule. Par exemple :

   ```
   2, 5, 12, 17, 21, 35
   ```

3. Dans le package **labo7**, ajoutez une classe nommée **Lotto**

4. Dans la classe **Lotto**, déclarez :

→ une fonction nommée **genererTirage** qui permet de créer un nouveau tableau contenant les numéros tirés :

```
static int[] genererTirage(int nbTirages, int numeroMax)
```

Le paramètre `nbTirages` indique le nombre de numéros à tirer (dans le cas du Lotto belge, ce paramètre aura la valeur 7)

Le paramètre `numeroMax` indique le plus grand nombre de la série (dans le cas du Lotto belge, ce paramètre aura la valeur 45)

Mettez en œuvre la logique du tirage expliquée précédemment. Utilisez la fonction *TableauEntiers.permuter*

A la fin du tirage, les numéros du tableau, à l'exception du numéro bonus situé à la dernière position, doivent être triés par ordre croissant à l'aide de la fonction *Arrays.sort*. Par exemple :

```
[2, 20, 26, 28, 38, 42, 30]
```

→ une fonction nommée **encoderGrille** qui permet de créer un nouveau tableau contenant les numéros de la grille de Lotto de l'utilisateur :

```
static int[] encoderGrille(int nbNumeros, int numeroMax)
```

Le paramètre `nbNumeros` indique le nombre de numéros que l'utilisateur doit encoder (dans le cas du Lotto belge, ce paramètre aura la valeur 6)

Le paramètre `numeroMax` indique le plus grand nombre de la série (dans le cas du Lotto belge, ce paramètre aura la valeur 45)

Si un numéro saisi par l'utilisateur n'est pas cohérent ou est déjà présent dans sa grille, alors recommencez l'acquisition de ce numéro. Utilisez la fonction *TableauEntiers.contient*

→ une fonction nommée **compterNumerosGagnants** qui permet de compter les numéros gagnants dans une grille de Lotto :

```
static int[] compterNumerosGagnants(int[] tirage, int[] grille)
```

La fonction retourne un tableau contenant deux éléments : le premier indique le nombre de numéros gagnants sans comptabiliser le numéro bonus (de 1 à 6) ; le second indique si le numéro bonus est obtenu (0 ou 1). Par exemple :

`[5, 1]` correspond à 5+

→ une fonction nommée **determinerRang** qui permet d'obtenir le rang correspondant au nombre de numéros gagnants :

```
static int determinerRang(int[] numerosGagnants)
```

[Astuce] Si *n* est le nombre de numéros gagnants (sans comptabiliser le numéro bonus), alors le rang peut être obtenu par la formule $(7 - n) \times 2 - 1$

→ une fonction nommée **obtenirGain** qui permet d'obtenir le montant du gain remporté par le joueur en fonction du rang :

```
static double obtenirGain(int rang)
```

Reprenez les montants listés dans le tableau en début d'énoncé

TEST UNITAIRE

Avant d'utiliser les fonctions précédemment définies, appliquez des tests à trois d'entre elles : **compterNumerosGagnants**, **determinerRang** et **obtenirGain**.

1. Dans le répertoire **test** du projet **prb**, ajoutez un package nommé **labo7**
2. Dans le package **labo7** du répertoire **test**, ajoutez une classe de test nommée **LottoTest**
3. Déclarez des fonctions de test permettant de valider le fonctionnement des trois fonctions précitées

   [Attention] Pour comparer le contenu de deux tableaux, utilisez l'assertion *assertArrayEquals*

PROGRAMME PRINCIPAL

1. Dans le package **labo7**, ajoutez une classe nommée **LoterieNationale** comportant une fonction **main**
2. Dans la fonction **main**, réalisez le programme tel qu'il est montré dans l'exemple d'exécution ci-dessous. Pour ce faire, utilisez les fonctions précédemment déclarées dans les classes **TableauEntiers** et **Lotto**

   Remarquez que le texte de l'option 1 du menu change selon que l'utilisateur a rempli ou non sa grille de Lotto

   Si le choix fait par l'utilisateur n'est pas cohérent, affichez le message *"Choix incorrect !"*

   Lorsque l'utilisateur choisit l'option 2, **un nouveau tirage du Lotto est effectué**. Si l'utilisateur a déjà rempli sa grille de Lotto, ses résultats sont affichés

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les données saisies par l'utilisateur sont représentées en vert.

```
Loterie Nationale
1. Remplir une grille de Lotto
2. Consulter mon gain pour le dernier tirage
3. Quitter
Choix ? 2

Le tirage est 2, 20, 26, 28, 38, 42, 30
Vous n'avez pas rempli de grille

Loterie Nationale
1. Remplir une grille de Lotto
2. Consulter mon gain pour le dernier tirage
```

```
3. Quitter
Choix ? 1

Numéro 1 ? 2
Numéro 2 ? 5
Numéro 3 ? 12
Numéro 4 ? 17
Numéro 5 ? 21
Numéro 6 ? 35
Votre grille est 2, 5, 12, 17, 21, 35

Loterie Nationale
1. Modifier ma grille de Lotto
2. Consulter mon gain pour le dernier tirage
3. Quitter
Choix ? 2

Le tirage est 4, 24, 25, 27, 37, 39, 40
Vous avez 0 numéro gagnant
Vous gagnez 0,00 euros

Loterie Nationale
1. Modifier ma grille de Lotto
2. Consulter mon gain pour le dernier tirage
3. Quitter
Choix ? 2

Le tirage est 17, 20, 21, 22, 34, 35, 1
Vous avez 3 numéros gagnants
Vous gagnez 5,00 euros

Loterie Nationale
1. Modifier ma grille de Lotto
2. Consulter mon gain pour le dernier tirage
3. Quitter
Choix ? 2

Le tirage est 1, 5, 10, 17, 30, 37, 35
Vous avez 2 numéros gagnants et le numéro bonus
Vous gagnez 3,00 euros

Loterie Nationale
1. Modifier ma grille de Lotto
2. Consulter mon gain pour le dernier tirage
3. Quitter
Choix ? 3

Fin du programme.
```