TP 07

TABLEAUX, BOUCLES ET CONDITIONS

Créer un package : « fr.algorithmie »

EXERCICE AFFICHAGEINVERSE

- Créer une classe **AffichageInverse**
- Soit le tableau suivant : int[] array = {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- Afficher l'ensemble des éléments du tableau grâce à une boucle
- Afficher l'ensemble des éléments dans l'ordre inverse du tableau
- Créer un tableau arrayCopy et copier tous les éléments de array dans arrayCopy

EXERCICE INVERSIONCONTENU

- Créer une classe **InversionContenu**
- Soit le tableau suivant : int[] array = {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- Créer un tableau arrayCopy et copier tous les éléments de array dans arrayCopy **mais dans l'ordre inverse**.
- Afficher l'ensemble des éléments des 2 tableaux

EXERCICE AFFICHAGEPARTIEL

- Créer une classe **AffichagePartiel**
- Soit le tableau suivant : int[] array = {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- **Combiner une boucle et un test** de manière à n'afficher que les entiers supérieurs à 3
- Combiner une boucle et un test de manière à n'afficher que les entiers pairs
- Combiner une boucle et un test de manière à n'afficher que les valeurs correspondant aux index pairs
- Combiner une boucle et un test de manière à n'afficher que les entiers impairs

Exercice RechercheMax

- Créer une classe **RechercheMax**
- Soit le tableau suivant : int[] array = {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- Rechercher le plus grand élément du tableau

Exercice RechercheMin

- Créer une classe **RechercheMin**
- Soit le tableau suivant : int[] array = {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- Rechercher le plus petit élément du tableau

Exercice CalculMoyenne

- Créer une classe **CalculMoyenne**
- Soit le tableau suivant : int[] array = {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- Quelle est la moyenne des éléments du tableau ?

Exercice SommeDeTableaux

- Créer une classe **SommeDeTableaux**
- {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- {-1, 12, 17, 14, 5, -9, 0, 18, -6, 0, 4, -13, 5, 7, -2, 8, -1} ;
- Créer un tableau qui contient la somme des 2 précédents tableaux

Exercice SommeDeTableauxDiff

- Créer une classe **SommeDeTableauxDiff**
- {1, 15, -3, 0, 8, 7, 4, -2, 28, 7, -1, 17, 2, 3, 0, 14, -4} ;
- {-1, 12, 17, 14, 5, -9, 0, 18 } ;
- Créer un tableau qui contient la somme des 2 précédents tableaux

Exercice ComparaisonTableau

- Créer une classe **ComparaisonTableau**
- Soit les tableaux suivants :
    - int[] array1 = {1, 15, -3, 8, 7, 4, -2, 28, -1, 17, 2, 3, 0, 14, -4} ;
    - int[] array2 = {3, -8, 17, 5, -1, 4, 0, 6, 2, 11, -5, -4, 8} ;
- Ecrire le code permettant de compter le nombre d'éléments en commun dans ces 2 tableaux

Exercice TriParSelection

- Créer une classe **TriParSelection**
- Soit le tableau suivant :
    - int[] array2 = {3, -8, 17, 5, -1, 4, 0, 6, 2, 11, -5, -4, 8} ;
- Implémenter la méthode tri par sélection

Exercice TriABulles

- Créer une classe **TriABulles**
- Soit le tableau suivant :
    - int[] array2 = {3, -8, 17, 5, -1, 4, 0, 6, 2, 11, -5, -4, 8} ;
- Implémenter la méthode de tri à bulles

Exercice FirstLast6

- Créer une classe **FirstLast6**
- Dans cette classe, on déclare un tableau d'entiers
- On calcule une valeur booléenne qui contrôle le tableau de la sorte :
    - elle vaut true si le tableau a au moins 1 élément et si le premier élément ou le dernier élément vaut 6.
    - elle vaut false dans les autres cas
- écrire l'algo de valorisation de cette variable avec le minimum de ligne

Exercice FirstLast

- Créer une classe **FirstLast**
- Dans cette classe, on déclare un tableau d'entiers
- On calcule une valeur booléenne qui contrôle le tableau de la sorte :
    - elle vaut true si le tableau est de longueur supérieure ou égale à 1 et que le premier et le dernier élément du tableau ont la même valeur
    - elle vaut false dans les autres cas
- écrire l'algo de valorisation de cette variable avec le minimum de ligne

Exercice Rotation (Difficile)

- Créer une classe **Rotation**
- Dans cette classe, on déclare un tableau d'entiers
- Effectuez une rotation à droite des éléments.
- Exemple : si vous avez {0,1,2,3} vous obtenez {3,0,1,2}

Exercice FabriquerMur (Difficile - Facultatif)

- Copiez la classe **FabriquerMur** dans votre projet STS
- Dans cette classe vous devez mettre au point la méthode **fabriquerMur**
- Cette méthode doit produire un algorithme qui retourne s'il est possible ou non de fabriquer un mur avec des briques de longueur 1 et des briques de longueur 5.
- Exemples :
    - j'ai 2 briques de longueur 1 et 2 briques de longueur 5, est-il possible de créer un mur de 11m ? la réponse est oui, il suffit de prendre 2 briques de 5 et une brique de 1.
    - j'ai 3 briques de longueur 1 et 1 brique de longueur 5, est-il possible de créer un mur de 9m ? la réponse est non.
- Veuillez compléter la méthode fabriquerMur qui prend en paramètres :
    - nbSmall : le nombre de briques de longueur 1
    - nbBig : le nombre de briques de longueur 5
    - longueur : la taille du mur.
- A l'exécution les méthodes **verifier** exécutées avec diverses valeurs de paramètres permettent de dire si oui ou non votre algorithme fonctionne.

```
public class FabriquerMur {

	public static void main(String[] args) {

		// Tests de vérification
		verifier(3, 1, 8, true);
		verifier(3, 1, 9, false);
		verifier(3, 2, 10, true);
		verifier(3, 2, 8, true);
		verifier(3, 2, 9, false);
		verifier(6, 1, 11, true);
		verifier(6, 0, 11, false);
		verifier(1, 4, 11, true);
		verifier(0, 3, 10, true);
		verifier(1, 4, 12, false);
		verifier(3, 1, 7, true);
		verifier(1, 1, 7, false);
	}

	static boolean fabriquerMur(int nbSmall, int nbBig, int longueur) {
		boolean resultat = false;
```

```
                // TODO: implémenter l'algo

                return resultat;
        }

        private static void verifier(int nbSmall, int nbBig, int longueur, boolean b) {
                if (!fabriquerMur(nbSmall, nbBig, longueur) == b) {
                        throw new RuntimeException("Test (" + nbSmall + ", " + nbBig + ", " +
longueur + ") NON passant.");
                }
        }
}
```

EXERCICE INTERACTIFTANTQUE

Créer une classe **InteractifTantQue**

Ecrire un programme qui demande un nombre à l'utilisateur qui doit être obligatoirement compris entre 1 et 10 :

- Tant que ce nombre n'est pas compris entre 1 et 10, le programme redemande un nombre à l'utilisateur.
- Si le nombre est compris entre 1 et 10, le programme affiche ce nombre et se termine.

Instruction pour poser une question à l'utilisateur :

Nous allons utiliser la classe java.util.Scanner.

```
Scanner scanner = new Scanner(System.in) ;
int nb = scanner.nextInt() ;
```

EXERCICE INTERACTIFTABLEMULT

Créer une classe **InteractifTableMult**

Ecrire un programme qui demande un nombre à l'utilisateur **qui doit** être compris entre 1 et 10. Une fois que le nombre est bien entre 1 et 10, le programme affiche la table de multiplication de ce nombre. Exemple :

Table de 3 :

3 * 1 = 3

3 * 2 = 6

…

3 * 10 = 30

Exercice InteractifChiffresSuivants

Créer une classe **InteractifChiffresSuivants**

Ecrire un programme qui demande un nombre à l'utilisateur puis qui affiche les 10 nombres suivants. Par exemple si l'utilisateur saisit 5, le programme affiche : 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.

Exercice InteractifSommeArithmetique

Créer une classe **InteractifSommeArithmetique**

Ecrire un programme qui demande un nombre à l'utilisateur puis calcule la somme de tous les entiers compris entre 1 et ce nombre.

Exemple si l'utilisateur saisit 5, le programme affiche: 15

Exercice InteractifPlusGrand

Créer une classe **InteractifPlusGrand**

Ecrire un programme qui demande 10 nombres à un utilisateur et qui affiche le plus grand de ces nombres.

Exercice InteractifPlusMoins

Créer une classe **InteractifPlusMoins**

Ecrire un jeu qui :

- choisit un nombre aléatoire entre 1 et 100
- puis demande à l'utilisateur de trouver ce nombre en lui indiquant s'il est au-dessus ou en dessous du nombre,
- Lorsque l'utilisateur a trouvé le nombre, le programme affiche « Bravo, vous avez trouvé en N coups » où N représente le nombre d'essais effecté par l'utilisateur
- le programme se termine.

Exercice InteractifStockageNombre (Difficile)

Créer une classe **InteractifStockageNombre**

Faire un programme avec le menu suivant :

1. Ajouter un nombre
2. Afficher les nombres existants.

Description :

Demander à l'utilisateur de choisir une option dans le menu.

Si l'utilisateur sélectionne l'option 1, le programme demande un nombre à l'utilisateur puis l'ajoute à un tableau.

Si l'utilisateur sélectionne l'option 2, le programme affiche le contenu du tableau.

Si le tableau est plein, écrire un algorithme pour agrandir le tableau.

EXERCICE INTERACTIFFIBONNACI (DIFFICILE)

Créer une classe **InteractifFibonacci**

La suite de Fibonacci est une suite qui commence par 0 et 1 et dans laquelle le **nombre** de rang **N** est égal à la somme des nombres de rangs N-1 et N-2

- Créer une classe TestFibonacci
- Demander à l'utilisateur de choisir un rang N
- Ecrire un algorithme qui calcule et affiche le nombre de rang N

EXERCICE INTERACTIF21BATONS (DIFFICILE)

Créer une classe **Interfactif21Batons**

Le jeu est simple mais la réalisation est plus délicate. Vous allez jouer contre l'ordinateur. Celui qui prend le dernier baton a perdu.

Dans ce TP vous allez devoir imaginer vous-même le mécanisme à mettre en place, sur la base de ce que vous avez vu précédemment.