Programmation Java @ Et3
Polytech Paris-Saclay | 2024-25

Nom : BRACQ
Prénom : Paolo

Exercice 2

Vous allez développer une classe utilitaire en Java permettant de réaliser divers traitements sur des tableaux d'entiers.

1. Créez une classe *au nom approprié* dans un package *pertinent*. Ajoutez une méthode statique à votre classe permettant de créer un tableau d'entiers (int[]) à partir d'un tableau de chaînes de caractères (String[]).

#2.1.1 code de la classe

```
package Users;

public class TableauUser {

    public static int[] charToInt(String[] charTableau) {
        if (charTableau == null || charTableau.length == 0) {
            return new int[0];
        }

        int[]intTableau = new int[charTableau.length];
        for (int i=0; i<charTableau.length; i++){
            intTableau[i] =Integer.parseInt(charTableau[i]);
        }
        return intTableau;
    }
```

#2.1.2 Pourquoi est-il utile d'associer notre classe à un *package* ?

```
On associe notre classe à un package pour optimizer notre code,
évite de potentiels conflits de noms, pour pouvoir controller les
accès et pour faciliter la reutilization des classes à l'avenir.
```

#2.1.3 Comment la méthode de conversion peut-elle se comporter en cas de problème de conversion, i.e. si une des chaînes du tableau ne peut pas être convertie en entier à l'exécution ? Essayez-le et commentez.

```
On n’a rien écrit de particulier dans la méthode qui permet de
gérer si une chaîne du tableau ne peut pas être convertie, on doit
donc avoir un message d’erreur.
J’ai créé une classe Test pour verifier cela:
package Users;

public class Tests {
    public static void main(String[] args) {
        String[] tableauInvalide = {"10", "20", "abc", "30"};
        int[] result
=Users.TableauUser.charToInt(tableauInvalide);
        System.out.println(result[2]);
    }
}

Et le résultat est:

Exception in thread "main" java.lang.NumberFormatException: For
input string: "abc"
     at
java.base/java.lang.NumberFormatException.forInputString(NumberFor
matException.java:67)
     at java.base/java.lang.Integer.parseInt(Integer.java:588)
     at java.base/java.lang.Integer.parseInt(Integer.java:685)
     at Users.TableauUser.charToInt(TableauUser.java:8)
     at Users.Tests.main(Tests.java:6)

Process finished with exit code 1
```

2. Ajoutez une méthode `main` à cette classe pour tester la méthode précédente en lui transmettant le tableau de paramètres lu depuis la ligne de commandes.

#2.2.1 méthode `main`

```
public static void main(String[] args){
    int[] tableauConvertie = charToInt(args);
    System.out.println("Tableau converti :");
    for (int num : tableauConvertie) {
        System.out.print(num + " ");
    }
}
```

3. Ajoutez une méthode de classe affichant le contenu d’un tableau d’entiers passé en paramètres.

#2.3.1 méthode ajoutée

```
public static void affichageTableau(int[] intTableau){
    System.out.print("Voici les éléments du tableau: \n ");
    if (intTableau == null || intTableau.length == 0){
        System.out.print("Le tableau est vide ");
```

```
    }else {
        for (int i = 0; i < intTableau.length; i++) {
            System.out.print(intTableau[i] + "  ");
        }
    }
}
```

#2.3.2 Que se passe-t-il au niveau de la machine virtuelle en cas de dépassement de lecture des éléments d'un tableau, par exemple dans la méthode précédente ? Pourquoi ceci n'a-t-il pas mené à une erreur de compilation ?

```
Lorsqu'un programme Java tente d'accéder à un élément d'un tableau
en dehors de ses limites, cela entraîne une exception d'exécution
et non une erreur de compilation. Une exception, comme
ArrayIndexOutOfBoundsException, est levée lors de l'exécution du
programme. C'est la JVM qui génère l'exception.
```

4. Ajoutez une méthode à la classe retournant la valeur maximale des valeurs d'un tableau d'entiers passé en paramètre. Pensez à une valeur adaptée dans le cas d'un tableau vide.

#2.4.1 méthode ajoutée

```
public static int valeurMax(int[] intTableau){
    if (intTableau == null || intTableau.length == 0) {
        return -1;
    }else{
        int max=intTableau[0];
        for (int i = 1; i < intTableau.length; i++) {
            if (intTableau[i]>max){
                max=intTableau[i];
            }
        }
        return max;
    }

}
```

#2.4.2 test de la méthode ajoutée

```
package Users;

public class Tests {
    public static void main(String[] args) {
        int[] result ={11,2,54,9,0,76,99,13};
        System.out.print("Le maximum de ce tableau est la valeur :
"+ Users.TableauUser.valeurMax(result));
    }
}

avec comme résultat:
```

```
Le maximum de ce tableau est la valeur : 99
Process finished with exit code 0


package Users;

public class Tests {
    public static void main(String[] args) {
        int[] result ={};
        System.out.print("Le maximum de ce tableau est la valeur :
"+ Users.TableauUser.valeurMax(result));
    }
}

avec comme résultat:

Le maximum de ce tableau est la valeur : -1
Process finished with exit code 0
```

5. On souhaite ajouter une méthode de classe qui permettrait d'échanger la valeur minimale d'un premier tableau d'entiers transmis en paramètre avec la valeur maximale d'un second tableau d'entiers.

#2.5.1 méthode ajoutée (si cela est possible, justifier sinon)

```
//On peut créer une telle méthode mais cela implique de mettre les
deux tableaux en paramètre.
// Car nous n'avons pas implémenté de tableau dans la classe
Tableau User comme nous avions pu le faire avec les
autamponneuses.

public static void echangerMinMax(int[] tableau1, int[] tableau2)
{
    if (tableau1 == null || tableau2 == null || tableau1.length ==
0 || tableau2.length == 0) {
        System.out.println("Erreur : un des tableaux est vide.");
        return; // il sert de break
    }

    // Trouver la valeur minimale du premier tableau et son index
    int minIndex1 = 0;
    int minValue = tableau1[0];
    for (int i = 1; i < tableau1.length; i++) {
        if (tableau1[i] < minValue) {
            minValue = tableau1[i];
            minIndex1 = i;
        }
    }

    // Trouver la valeur maximale du second tableau et son index
    int maxIndex2 = 0;
```

```
    int maxValue = tableau2[0];
    for (int i = 1; i < tableau2.length; i++) {
        if (tableau2[i] > maxValue) {
            maxValue = tableau2[i];
            maxIndex2 = i;
        }
    }
```

#2.5.2 test de la méthode ajoutée

```
package Users;

public class Tests {
    public static void main(String[] args) {
        int[] result ={11,2,54,9,0,76,99,13};
        int[] result2 ={1,48,120,39,7,19,80,47};
        Users.TableauUser.echangerMinMax(result,result2);
        System.out.print("Voici les tableaux après l'échange :
\n");
        Users.TableauUser.affichageTableau(result);
        Users.TableauUser.affichageTableau(result2);
    }
}

Et comme résultat:

Voici les tableaux après l'échange :
Voici les éléments du tableau:
 11  2  54  9  120  76  99  13
Voici les éléments du tableau:
 1  48  0  39  7  19  80  47

Process finished with exit code 0
```

6. On souhaite maintenant ajouter une méthode de classe qui retourne la somme des entiers transmis en acceptant un nombre quelconque d'entiers directement en paramètres de la méthode (c'est à dire non plus un tableau, mais un nombre de paramètres effectifs quelconque). Prévoir la situation où aucun entier ne serait transmis.

#2.6.1 méthode ajoutée

```
public static int SommeEntiers(int... entiers){
    if (entiers.length == 0){
        System.out.println("Aucun entier n'a été transmis.");
        return 0;  // Si aucun entier n'est transmis, on retourne
0
    }
    int result =0;
    for ( int num : entiers){
        result += num;
    }
```

```
    return result;
}
```

#2.6.2 test de la méthode ajoutée

```
public class Tests {
    public static void main(String[] args) {
        System.out.print("Voici la somme des entiers
11,2,54,9,0,76,99,13 : \n"+
Users.TableauUser.SommeEntiers(11,2,54,9,0,76,99,13)+" \n");
        System.out.print("Voici la somme d'aucun entier' : \n" );
        System.out.print(Users.TableauUser.SommeEntiers());
    }
}

Et le résultat:

Voici la somme des entiers 11,2,54,9,0,76,99,13 :
264
Voici la somme d'aucun entier' :
Aucun entier n'a été transmis.
0
Process finished with exit code 0
```

7. Testez votre classe depuis la méthode `main` d'une nouvelle classe d'un autre package de votre choix (en simulant donc l'utilisation de votre classe utilitaire par un code indépendant). Quels niveaux de visibilité pour les membres de votre première classe sont accessibles depuis la seconde ?

#2.7.1 code de la classe ajoutée

```
package TestTableauUser;

import Users.TableauUser; // Importer la classe TableauUser du
package Users

public class TestTableauUser {

    public static void main(String[] args) {
        // Test des méthodes de TableauUser

        // Exemple d'appel à la méthode charToInt()
        String[] argsTableau = {"1", "2", "3", "4"};
        int[] tableauConverti =
TableauUser.charToInt(argsTableau);

        // Affichage du tableau
        TableauUser.affichageTableau(tableauConverti);

        // Test de la méthode valeurMax()
        int max = TableauUser.valeurMax(tableauConverti);
        System.out.println("Valeur maximale : " + max);
```

```
        // Test de la méthode echangerMinMax() avec deux tableaux
        int[] tableau1 = {5, 3, 8};
        int[] tableau2 = {7, 1, 9};
        TableauUser.echangerMinMax(tableau1, tableau2);

        // Affichage des tableaux après échange
        TableauUser.affichageTableau(tableau1);
        TableauUser.affichageTableau(tableau2);

        // Test de la méthode SommeEntiers()
        int somme = TableauUser.SommeEntiers(1, 2, 3, 4);
        System.out.println("Somme des entiers : " + somme);
    }
}
```

Alors si je n'importe pas la classe je ne peux faire aucun test, ce qui est logique. Par contre, ayant choisi au début du TP de mettre toutes mes méthodes en public et bien je n'ai aucun problème à les utilizer depuis un package different. J'ai fait ce choix car les méthodes que nous avons codé sont des méthodes utiles à tous.