Initiation au développement - SDA : TP2 (Deux séances)

DUT/INFO/R1_01
version 2021-2022 (PN BUT 2021)

Table des matières

1. Environnement de travail
  - 1.1. Projet et classe(s)
  - 1.2. Organisation des sous-programmes
2. Exercice 1 : enregistrement TNP
  - 2.1. Définition du type enregistrement
  - 2.2. Constructeur
3. Exercice 2 : Procédure saisieTableau(·)
4. Exercice 3 : Procédure afficherTableau(·)
5. Exercice 4 : Fonction maxST(·)
6. Exercice 5 : Procédure triMax(·)
7. Exercice 6 : Procédure sauverTableau(·)
  - 7.1. Définition du sous-programme
  - 7.2. Appel du sous-programme
8. Exercice 7 : Procédure chargerTableau(·)
  - 8.1. Définition du sous-programme de chargement
  - 8.2. Appel du sous-programme de chargement
  - 8.3. Retour sur la sauvegarde
9. Aller plus loin
10. Avant de partir

1. Environnement de travail

1.1. Projet et classe(s)

Tout comme dans les TPs d'IAP, réaliser les actions suivantes pour configurer votre environnement de travail.

1. Lancer BlueJ.
2. Créer et nommer (nous vous suggérons TP2_NumeroDeGroupe_Nom_Prenom) un nouveau projet où vous souhaitez placer vos sources java.

1.2. Organisation des sous-programmes

1. Créer une classe **LibrairieTri**. Nous allons structurer cette classe comme nous commençons à en prendre l'habitude. La classe va contenir :
   - des sous-programmes, c'est-à-dire un ensemble de fonctions et de procédures qui seront disponibles à qui voudra, à l'intérieur de la classe
   - une fonction qui jouera le rôle de fonction principale, c'est-à-dire qui sera le poste de commandement. Elle enchaînera des appels aux sous-programmes précédemment écrits. On pourra l'écrire en tant que fonction principale Java (elle devra, pour être reconnue comme telle, s'appeler `main`, ne rien retourner, et avoir un unique paramètre de type tableau de `String`), ou en tant que sous-programme habituel.

     Ici, nous n'avons pas besoin que Java reconnaisse la fonction principale comme telle, donc nous désignons la fonction **`principale()`**, sans argument et ne renvoyant rien, comme la fonction principale.
2. Mais commençons d'abord par créer le type tableau non plein dans le premier exercice.

2. Exercice 1 : enregistrement **`TNP`**

2.1. Définition du type enregistrement

Inspirez-vous du TD pour écrire une classe `TNP` (dans le fichier `TNP.java`) correspondant à un type enregistrement qui représente des tableaux non plein de réels.

2.2. Constructeur

Dans le TD, nous avions laissé de côté les constructeurs de ce type enregistrement. Écrire un constructeur qui a besoin d'un seul paramètre : le nombre maximum d'éléments qui peuvent être stockés dans le tableau. Vous remplirez les champs en adéquation.

Notez bien la différence entre le nombre d'éléments que l'on peut stocker au maximum, et le nombre d'éléments qui sont significatifs dans le tableau. Ici, il s'agit de réserver de la place, mais de garder l'information selon laquelle aucune case n'a de valeur significative.

3. Exercice 2 : Procédure **`saisieTableau(·)`**

1. Ecrire en **Java** la procédure **`saisieTableau(·)`** dont voici la description :

```
  /**
   *  Lit au clavier pfNbElements réels et les place
dans le TNP pfTabR
   *
   * @param pfNbElements IN : le nombre de valeurs à
faire saisir à l'utilisateur
   * @param pfTabR OUT : le TNP
   *
   * Préconditions ?
   */
```

1. Compléter la fonction **`principale()`** de la classe **LibrairieTri** pour tester votre procédure : construire un tableau non plein qui réserve 100 cases en mémoire, puis faire saisir 10 valeurs à l'utilisateur dans ce tableau non plein. À l'aide du debbuger, vérifier ce que contient le tableau non plein donné en paramètre.

4. Exercice 3 : Procédure **`afficherTableau(·)`**

Écrire en **Java** la procédure `afficherTableau(·)` dont voici la description :

```
  /**
   *  Affiche à l'écran les valeurs significatives du
TNP pfTabR
   *
   * @param pfTabR IN : le tableau non plein
   *
   */
```

Tester la procédure en en faisant un appel dans la fonction **`principale`**.

5. Exercice 4 : Fonction **`maxST(·)`**

Ecrire en **Java** la fonction **`maxST(·)`** vue en TD, dont voici pour rappel les caractéristiques :

```
  /**
   *  permet de déterminer le rang de la valeur
maximum des éléments du tableau
   * non plein de réels pfTabR compris entre les
indices pfDeb et pfFin
   *
   * @param pfTabR IN : le tableau non plein des
valeurs réelles
   * @param pfDeb IN : indice de début de sous-
tableau
   * @param pfFin IN : indice de fin de sous-tableau
   *
   * @return l'indice du maximum dans le sous-tableau
délimité par pfDeb et pfFin (compris)
   *
   * Préconditions : 0 <= pfDeb <= pfFin <
pfTabR.nbElt
   */
```

Glossaire V.L.

Identificateur	Type	Rôle
`i`	Entier	Indice du tableau
`rangMax`	Entier	Valeur de retour

Algorithme :

```
DEBUT
    /* initialisation de l'indice du maximum */
    rangMax ← pfDeb
    POUR (INIT : i ← pfDeb + 1; TEST : i <= pfFin
; MAJ : i ← i + 1 ) FRE
        SI pfTabR.lesElements[i] >
pfTabR.lesElements[rangMax] ALORS
            rangMax ←  i
        FINSI
    FINPOUR
    retourne rangMax
FIN
```

Indications techniques

1. Vous placerez la fonction **`maxST(·)`** dans la classe **LibrairieTri**
2. Pour éviter de devoir saisir un tableau de test à chaque fois, Vous créerez une fonction **creerTNPTest** qui renvoie un tableau non plein de test, qui comporte 5 valeurs significatives :

```
    public static TNP creerTNPTest() {
        TNP tab = new TNP(7) ;
        tab.lesElements[0] = 1.0 ;
        tab.lesElements[1] = 2.0 ;
        tab.lesElements[2] = 3.0 ;
        tab.lesElements[3] = 4.0 ;
        tab.lesElements[4] = 5.0 ;
        tab.lesElements[5] = 7.3 ;
        tab.lesElements[6] = 5.2 ;
        tab.nbElt = 5 ;
          return tab;
    }
```

1. Vous **testerez** votre fonction en ajoutant les cas d'un jeu d'essais significatif dans la fonction **`principale()`** de la classe **LibrairieTri**, en vous inspirant éventuellement de

```
        TNP tabTest = creerTNPTest() ;
        if (maxST(tabTest, 1, 1) == 1)
    {
          System.out.print(".") ;
        } else {
          System.out.print("F") ;
        }
```

où l'on teste un cas simple de **`maxST`** (on vérifie que le rang du maximum entre les indices 1 et 1 est bien 1 selon **`maxST`** !).

6. Exercice 5 : Procédure **`triMax(·)`**

1. Écrire en **Java** la procédure **`triMax(·)`** vue en TD (et qui utilise **`maxST(·)`**).
2. Compléter la fonction **`principale()`** de la classe pour tester la procédure **`triMax(·)`** en utilisant les procédures écrites jusqu'ici.

7. Exercice 6 : Procédure **`sauverTableau(·)`**

Il arrive souvent de vouloir qu'un tableau survive à l'exécution d'un programme. Si on utilise le tableau seulement à travers une variable de type tableau (ou tableau non plein), cette variable, et donc le contenu du tableau, disparaissent à la fin de l'exécution.

Pour pouvoir récupérer les valeurs d'un tableau après la fin d'une exécution, on peut les stocker dans un fichier.

7.1. Définition du sous-programme

Ecrire en **Java** la procédure **sauverTableau(·)** dont voici la description :

```
  /**
   * Enregistre les pfNbElements réels du tableau non
plein pfTabR dans le fichier
   * de nom pfNomFichier
   *
   * @param pfTabR IN : un tableau non plein de
valeurs réelles
   * @param pfNomFichier IN : le nom du fichier cible
   *
   */
```

Indications techniques

- Pour sauvegarder des informations dans un fichier, il suffit de remplacer le désormais classique **`System.out`** par le nom d'une variable désignant un fichier **ouvert en écriture**.

```
import java.io.*;

//...

    PrintStream out = new
PrintStream("monFichier.txt");   /*❶
*/
    out.println("3,14");
    out.println(33);
    out.println("Hello Disque");
    out.close();   /*❷*/
```

  ❶ out est un descripteur de fichier ouvert par PrintStream() en écriture

  ❷ après fermeture du descripteur, il n'est plus possible d'écrire dans le fichier

- Vous allez rencontrer un problème d'exception. Pour rappel, lorsqu'un sous-programme ou un constructeur est déclaré comme pouvant soulever une exception, il faut s'en occuper. Deux façons de le faire : soit l'attraper, soit la laisser passer. Ici, vous l'attraperez et l'afficherez à l'écran si elle se manifeste. Pour plus d'informations, se référer au TP d'IAP correspondant.

7.2. Appel du sous-programme

Ajouter un appel à **`sauverTableau`** dans la fonction **`principale()`** de la classe **LibrairieTri**, et examiner le contenu du fichier dans lequel vous avez stocké les valeurs.

8. Exercice 7 : Procédure **`chargerTableau(·)`**

8.1. Définition du sous-programme de chargement

Écrire en **Java** la procédure `chargerTableau(·)` dont voici la description

```
  /**
   * Lit pfNbElements réels dans le fichier de nom
pfNomFichier et les place
   * dans le tableau non plein pfTabR
   *
   * @param pfTabR OUT : un tableau non plein de
valeurs réelles
   * @param pfNbElements IN : le nombre de valeurs
   * @param pfNomFichier IN : le nom du fichier cible
   *
   * Préconditions A compléter
   */
```

Indications techniques

- Pour lire des informations dans un fichier, il suffit d'initialiser un *lecteur* avec le fichier ouvert en lecture ainsi :

```
    Scanner lecteur = new Scanner(new
File("monFichier.txt"));    /*❶*/
    variableDouble =
lecteur.nextDouble();
    variableInt = lecteur.nextInt();
    variableString = lecteur.next();
/*❷*/
    lecteur.close();    /*❸*/
```

❶ **`lecteur`** est un descripteur de fichier ouvert en lecture par **`Scanner()`**

❷ lecture du mot suivant

❸ après fermeture du descripteur il n'est plus possible de lire dans le fichier
  - Traiter les exceptions comme précédemment.

8.2. Appel du sous-programme de chargement

On pourrait s'attendre à ce que l'on puisse maintenant sauvegarder le contenu d'un tableau non plein dans un fichier, puis charger ce fichier dans un tableau non plein, et constater que ce tableau est le même que le tableau non plein initial. Essayez.

Que se passe-t-il ?

8.3. Retour sur la sauvegarde

Modifier votre fonction **`sauverTableau(·)`** pour qu'elle utilise le format de langue de votre machine (certainement *fr*) pour l'écriture des nombres.

Indications techniques

Pour formatter un nombre décimal selon la norme française, il faut le convertir lors de l'écriture de sa valeur ainsi :

```
import java.util.*;
import java.text.*;

...

    NumberFormat fr =
NumberFormat.getInstance(Locale.FRENCH);
// ❶

System.out.println(fr.format(111.12));
// ❷
```

❶ **`fr`** contient la description du format des nombres pour la France.

❷ **`fr.format(valeur)`** retourne une chaine de caractères représentant *valeur* au format Français.

Réessayez maintenant dans la fonction **`principale()`** de sauvegarder dans un fichier, puis de charger le fichier.

9. Aller plus loin

Écrire un ou des sous-programmes prenant un tableau non plein, triant les éléments pairs par ordre croissant dans la partie gauche et les éléments impairs par ordre décroissant dans la partie droite.

Par exemple, {1,4,6,23,49} doit être modifié en {4,6,49,23,1}.

10. Avant de partir

1. Enregistrer vos programmes sous **webetud2**.
2. N'oubliez pas de vous déconnecter.

Version 2021-2022 (PN BUT 2021)
Dernière mise à jour 2021-09-03 10:19:29 CEST