TP4 : REMPLACEMENT DE L'EXAMEN FINAL

SESSION HIVER 2020

CONSIGNES POUR LE TRAVAIL

1. Ce travail est divisé en cinq (5) questions.
2. Pour chaque question :
    a. Il y a une description de la tâche plus loin dans cet énoncé.
    b. Vous devrez écrire le code pour une méthode.
        i. Chaque méthode à écrire à des consignes spécifiques et différentes.
        ii. Un fichier contient les tests unitaires pour cette méthode (`TP4Test.java`).
    c. Une tâche complémentaire est assignée à chaque méthode et décrite pour chaque question.
    d. Chaque question est notée sur 5 points :
        i. 3 points pour la fonctionnalité du code (les tests unitaires fonctionnent).
        ii. 2 points pour la tâche complémentaire.
3. Des fichiers de code sont disponibles sur Moodle, vous devez placer vos réponses dans ces fichiers.
    a. `ArbreBinaire.java` : pour la question 5.
    b. `ExceptionPileVide.java` : pour la question 4.
    c. `Pair.java` : pour la question 5.
    d. `Phrase.java` : pour la question 3.
    e. `Pile.java` : pour la question 4.
    f. `Principali20.java` : pour les questions 1 et 2.
    g. `TP4Test.java` : contiens les tests unitaires pour les questions 1, 2, 3, 4 et 5.
4. Vous devez remettre une archive contenant ces fichiers.
    a. Remise sur Moodle.
    b. Le 3 mai 2020, avant 23 :55.
5. C'est un travail INDIVIDUEL.
6. S.V.P. : Si vous avez des questions, posez-les par courriels. Si je juge l'information pertinente, alors je placerai ma réponse dans le forum.

QUESTIONS

1. TRI DOUBLE SÉLECTION

- Utilise :
  - Le fichier `Principali20.java`.
- Où place la réponse :
  - La méthode `triDoubleSelection` et ses commentaires dans le fichier `Principali20.java`.
- Tâches :
  - (3 points) Écrire le code pour la méthode `triDoubleSelection`.
  - (2 points) Analyse de complexité temporelle de la méthode.

Cette méthode tri un tableau. Nous voulons utiliser un algorithme similaire au tri sélection. Dans le tri sélection normale, la boucle interne trouve le minima et ensuite l'échange avec le premier élément. Pour notre version, nous voulons profiter de la boucle interne qui parcourt le tableau pour trouver le minima et le maxima dans la même boucle. Ainsi, lorsque la boucle interne termine, nous pourrons placer 2 valeurs plutôt qu'une seule : le minima au début et le maxima à la fin. Cela va permettre à la boucle externe de faire moins de tours de boucle. Ajuster la boucle externe en conséquence. Pour ce code vous devez utiliser la méthode '`compareTo`' afin de comparer les éléments du tableau. Par contre, vous ne pouvez pas utiliser aucun autre appel de méthode. Vous pouvez utiliser toutes les instructions Java (`if, while, for, …`).

Tâche complémentaire. Vous devez faire une analyse de complexité temporelle pour l'algorithme que vous avez écrit pour le pire cas. Placez votre analyse en commentaire de la méthode `triDoubleSelection`. Placez les étapes de votre raisonnement dans cette analyse, **pas** seulement la réponse.

2. CLASSER

- Utilise :
  - Le fichier `Principali20.java`.
- Où place la réponse :
  - La méthode `classer` et ses commentaires dans le fichier `Principali20.java`.
- Tâches :
  - (3 points) Écrire le code pour la méthode `classer`.
  - (2 points) Exemple d'expression lambda.

Cette méthode déplace les éléments du tableau. Elle va placer tous les éléments qui la même caractéristique ensemble (consécutivement) dans le tableau. Par exemple, si nous avons le tableau [0,1,2,3,4], nous pourrions demander que ces éléments soient classés selon leur parité. Donc les valeurs paires ensemble et les valeurs impaires ensemble. Ce qui modifierait l'ordre des éléments et donnerait le tableau [0,2,4,1,3] par exemple. Il est à remarquer que cela pourrait aussi donner le tableau [1,3,0,2,4] ou [2,0,4,3,1]. La seule chose importante dans ces résultats : tous les éléments pairs se suivent et tous les éléments impairs se suivent.

Il reste permettre à l'utilisateur de notre méthode de nous donner la caractéristique de classement qu'il veut utiliser pour son tableau. Pour cela, nous lui demandons de nous donner une `Function`. Cette `Function` va permettre de transformer les valeurs du tableau en d'autres valeurs. Dans notre exemple, l'utilisateur pourrait nous donner une fonction qui calcule le reste de la division par 2. Ce qui transformerait le tableau [0,1,2,3,4] en tableau [0,1,0,1,0]. Ensuite, il reste à replacer les éléments du tableau selon ces valeurs. Toutes les valeurs qui ont donné 0 ensemble et toutes les valeurs qui ont donné 1 ensemble. Comme nous ne connaissons pas a priori le type du résultat de cette `Function`, nous devons utiliser `equals` pour les comparaisons entre les résultats de l'application de cette fonction.

Pour ce code, je vous conseille de réutiliser le code du tri insertion. Cela permet d'insérer les éléments à leur place. Vous pouvez utiliser `length` pour la taille des tableaux et les méthodes `apply` et `equals`. Aucune autre méthode ne peut être utilisée. Vous pouvez utiliser toutes les instructions Java (`if, while, for, …`).

Tâche complémentaire. Un client à un tableau de chaîne de caractères. Il voudrait regrouper les chaînes qui ont 5 caractères ou plus ensemble et qui commencent par une majuscule ensemble. Les autres chaînes seraient dans l'autre groupe (moins que 4 caractères ou commençant par une minuscule). Vous devez construire la fonction (f) de classification qui permettrait d'utiliser la méthode `classer` pour classer les chaînes de caractères dans le tableau.

Par exemple, l'appel à la méthode `classer` suivant :

```
String [] t = { "allo", "Comment", "Ca", "l", "universite" };
classer( t, f );
```

Pourrait donner le tableau `{ "Comment", "allo", "Ca", "l", "universite" }`.

Le mot **`Comment`** est le seul à avoir plus de 4 caractères et commençant par une lettre majuscule. Il est donc placé au début. Il est à remarquer qu'il aurait pu être placé à la fin.

Pour la question complémentaire, vous devez placer le code de la fonction (f) en commentaire de la méthode `classer`. Votre fonction doit être écrite en utilisant la notation Lambda de Java. Pour cette tâche complémentaire, vous pouvez utiliser toutes les méthode standard de Java.

3. NETTOYER

- Utilise :
  - Le fichier `Phrase.java`.
- Où place la réponse :
  - La méthode `nettoyer` dans le fichier `Phrase.java`.
- Tâches :
  - (3 points) Écrire le code pour la méthode `nettoyer`.
  - (2 points) Optimisation.

La classe `Phrase` est une Structure de Données qui est utile pour contenir les mots d'une `Phrase`. Elle est réalisée en utilisant l'héritage de la classe `ArrayList`. Plus particulièrement, un `ArrayList` de `String`. Chaque `String` est utilisé pour représenter un mot.

Lorsque nous analysons les phrases (analyse de similarité, comme dans le TP3), il arrive que nous voulions enlever des mots moins significatifs (appelé 'mot vide') de cette phrase avant de faire l'analyse. La classe `Phrase` contient un tableau statique contenant une liste de mots vide.

Vous devez écrire le code de la méthode nettoyer qui va enlever les mots vides de la `Phrase` courante (`this`). Tous les mots vides doivent être enlevés, mais les mots restants doivent conserver leurs ordres relatifs. C'est-à-dire, si un mot venait avant un autre mot dans la phrase de départ, il doit encore être avant dans la phrase nettoyée.

Votre code peut utiliser toutes les méthodes de la classe `ArrayList`, vous pouvez aussi utiliser les méthodes `equals`, `compareTo`, et `compareToIgnoreCase`, mais aucune autre. Vous pouvez utiliser toutes les instructions de Java (`if, while, for, …`).

Tâche complémentaire. Nous avons vu en classe un algorithme qui permettrait d'accélérer de beaucoup cette méthode. Les 2 points de la tâche supplémentaire seront donnés à ceux qui utiliseront cet algorithme.

4. ÉCHANGER

- Utilise :
  - Le fichier `Pile.java`.
  - Le fichier `ExceptionPileVide.java`.
- Où place la réponse :
  - La méthode `echanger` et ses commentaires dans le fichier `Pile.java`.
- Tâches :
  - (3 points) Écrire le code pour la méthode `echanger`.
  - (2 points) Expliquer votre algorithme.

Vous devez écrire le code pour la méthode échanger de la classe `Pile`. Cette `Pile` utilise une liste chaînée. Votre méthode doit échanger le premier `Chainon` (au sommet) de la pile avec la deuxième `Chainon` (celui juste en dessous dans la pile). Ce n'est pas seulement les valeurs qui changent de place, mais plutôt les `Chainon` s au complet. L'objectif de cette question est de vous faire manipuler les références.

Si la pile est vide ou si la pile ne contient qu'un seul élément, alors la méthode ne fait rien. C'est seulement s'il y a deux éléments ou plus que la méthode fait l'échange.

Pour ce code, vous ne pouvez utiliser que les instructions Java (`if, while, for, …`). Vous ne pouvez pas utiliser aucune méthode.

Tâche complémentaire. Dans les commentaires de la méthode `echanger`, vous devez expliquer votre algorithme. Cette description doit être complète. Imaginez que vous expliquez la méthode à des étudiants. Expliquez correctement le fonctionnement de votre code. Cette explication devrait prendre de 10 à 15 lignes d'aux maximum 80 caractères.

5. HAUTEUR

- Utilise :
  - Le fichier `ArbreBinaire.java`.
  - Le fichier `Pair.java`.
- Où place la réponse :
  - La méthode `hauteur` et ses commentaires dans le fichier `ArbreBinaire.java`.
- Tâches :
  - (3 points) Écrire le code pour la méthode `hauteur`.
  - (2 points) Expliquer votre algorithme.

La hauteur d'un Nœud dans un arbre binaire de recherche est définie (récursivement) de la façon suivante :

- Si le Nœud n'a pas d'enfant, alors la hauteur du Nœud est de 1.
- Si le Nœud a un enfant, alors la hauteur du Nœud est la hauteur de son enfant auquel nous ajoutons 1.
- Si le Nœud a deux enfants, alors nous prenons la hauteur du Nœud le plus haut (maximum) et lui ajoutons 1.

Vous devez écrire le code pour la méthode hauteur de la classe `ArbreBinaire`. Cette méthode accepte en argument une valeur cible. Vous devez trouver le `Nœud` contenant cette valeur et ensuite calculer la hauteur de ce `Nœud`. La méthode retourne la hauteur du `Nœud`. Si la valeur cible n'est pas dans l'arbre, alors la méthode retourne 0.

Pour ce code vous pouvez utiliser toutes les méthodes présentes dans la classe `ArbreBinaire` et la classe `Nœud` (la classe `Pair` n'a pas de méthode, mais vous pouvez l'utiliser). Vous pouvez aussi en ajouter dans ces classes. Vous pouvez utiliser toutes les instructions Java (`if, while, for, …`). Votre code peut être récursif, mais ce n'est pas obligatoire.

Tâche complémentaire. Dans les commentaires de la méthode `hauteur`, vous devez expliquer votre algorithme. Cette description doit être complète. Imaginez que vous expliquez la méthode à des étudiants. Expliquez correctement le fonctionnement de votre code. Cette explication devrait prendre de 10 à 15 lignes d'aux maximum 80 caractères.