BPOO - Sujet TD 4

Dut/Info-S2/M2103

Table des matières

PreReq	Cours 1 : approche objet. TD1 (références, tableaux). TD3 classes objets-encapsulation.
ObjTD	**Écrire une classe avec tableaux dynamiques - Encapsulation**.
Durée	**1 séance** de 1,5h

1. Principes de stockage de données

1.1. Stockage de données en mémoire / Possibilités du langage

Pour stocker des données, ce qui est donné comme disponible au programmeur en Java est :

1. les types de base,
2. le type tableau d'éléments d'un type donné,
3. le type "classe(/structure)" : des objets en mémoire contenant des attributs.

Pour stocker en mémoire "plusieurs" données, disons une "suite"/"séquence"/"liste"/… de données, il existe deux structures de programmation de base :

1. le tableau,
2. la structure chaînée (sous forme linéaire ou arborescente).

1.2. Structures de données

En dehors de tout contexte particulier (langage et application), une structure de donnée définit :

1. un ensemble d'informations organisées,
2. l'organisation de ces informations,
3. les opérations applicables sur ces informations … et leur résultat respectif.

Exemples: pile, file, liste, ensemble, map (clef/valeur), arbre, graphe, …

Pour les utiliser, il faut **construire/implémenter** ces structures de données à partir des possibilités de chaque langage.

Une implantation d'une structure de donnée repose sur :

- le choix d'une représentation mémoire,
- le choix d'une implémentation algorithmique des opérations de la structure de donnée.
- deux éléments fondamentaux à prendre en compte : espace mémoire - temps exécution/accès.

Dans les environnements de développement, on trouve des bibliothèques (packages java) qui offrent diverses implémentations des

structures de données.

Par exemple, en java on trouve au moins deux implémentations de la structure de donnée liste itérative :

- ArrayList<E> : liste implantée sous forme de tableau,
- LinkedList<E> : liste implantée sous forme de structure doublement chaînée.

On trouvera aussi :

- Structure de donnée pile :
  - Stack<E> : pile implantée sous forme de tableau,
  - et d'autres …
- Structure de donnée file :
  - ArrayDeque<E> : file implantée sous forme de tableau.
  - et d'autres …
- Structure de donnée ensemble (collection sans doublons)
  - HashSet<E> : ensemble implanté sous forme de paires clef/valeur,
  - TreeSet<E> : ensemble implanté sous forme d'arbre balancé.
- Structure de donnée map (association clef/valeur)
  - HashMap<K,V> : map implantée sous forme d'une table de hachage,
  - TreeMap<K,V> : map implantée sous forme d'arbre balancé.

---

2. But du TD

Étudier comment implémenter des "tableaux dynamiques" ou "comment changer la taille d'un tableau sans perdre les éléments". Implémenter une pile de String avec un tableau dynamique. Permet d'illustrer UN mode d'implémentation d'une structure de donnée.

---

3. Principe d'un "tableau dynamique"

Fonctionnement des tableaux en java :

- Ils sont créés au run-time.
- Une fois créés : la taille ne peut pas être modifiée.
- Une variable tableau contient une **référence** du tableau (pas le tableau lui-même).

Exercice : comment "agrandir/rétrécir" un tableau ?

Dans le programme joint (cf. document de réponse), indiquer le code nécessaire à ajouter pour agrandir le tableau référencé par `tabCourant` :

- `tabCourant` fait 5 éléments au départ,
- on y rentre 5 valeurs en <1>,
- il faudra agrandir `tabCourant` de `nbValSuppl` éléments,
- à la fin il faudra afficher les 5 premières valeurs saisies en <1> et les `nbValSuppl` valeurs saisies en <2>.

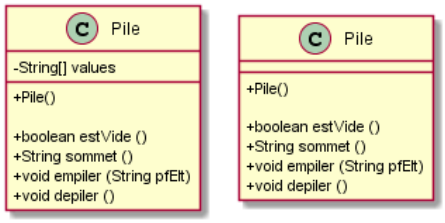
---

4. Classe à développer : Pile de String

La pile :

- Structure de donnée de type LIFO (Last In, First Out).
- Dernier élément "entré" dans la pile sera le premier "sorti".

On souhaite définir une classe **Pile de String** dont l'implémentation est un tableau "dynamique" de la taille exacte du nombre d'éléments contenus dans la Pile. L'implantation proposée ici consiste à changer le tableau de stockage à chaque modification de la pile.

Diagramme UML complet (gauche) et interface (droite) de la classe Pile	**Méthodes**
	Les méthodes : * `Pile()` : Constructeur par défaut qui construit une pile vide (sans éléments). * `boolean estVide ()` : retourne true si la pile est vide, false sinon. * `String sommet ()` : donne accès au sommet de la pile, l'élément en "haut" de la pile (le dernier empilé). ⇒ Précondition : sommet() valide ssi pilevide() == false. Lève une exception PileException si non respecté. * `void empiler (String pfElt)` : permet d'ajouter un élément `pfElt` en "haut" de la pile. * `void depiler ()` : enlève l'élément en "haut" de la pile. ⇒ Précondition : depiler() valide ssi pilevide() == false. Lève une exception PileException si non respecté.

NB : noter que dans certains cas (livres, sites web, etc.), l'en-tête de depiler() pourrait être **`String`** `depiler ()`, signifiant que depiler() pourrait ET enlever le sommet de la pile ET renvoyer la valeur de ce dernier.

4.1. Implémentation mémoire

Donner l'implémentation d'un objet Pile que l'on a créé avec une variable `maPile` NB : on ne représentera pas le contenu des éléments du tableau de la Pile qui sont des String

1. Avec un élément.
2. Après empilement d'un nouvel élément
3. Après empilement d'un nouvel élément
4. Après dépilement d'un élément
5. Après dépilement d'un élément
6. Après dépilement d'un élément

Java autorise de créer des tableaux de longueur 0 : un objet tableau qui existe et peut contenir 0 éléments (l'attribut `length` vaut 0).

4.2. Codage

Écrire les méthodes de la classe Pile implémentée par un tableau "dynamique" de la taille exacte du nombre d'éléments contenus dans la Pile.

En cas d'appel des méthodes hors préconditions, les méthodes lèveront une exception de type `PileException`.

Java autorise de créer des tableaux de longueur 0 : un objet tableau qui existe et peut contenir 0 éléments (l'attribut length vaut 0).

Dernière mise à jour 2017-02-16 16:45:58 CET