BPOO - Sujet TD 4

Dut/Info-S2/M2103

Table des matières

PreReq	Cours 1 : approche objet. TD3 classes objets -encapsulation.
ObjTD	**Ecrire 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.

C'est tout …

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

1. le tableau,

2. la structure chaînée (vu plus tard).

1.2. Structures de données

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, files, listes, ensembles, map (clef/valeur), arbres, graphes, …

Il faut **construire/implémenter** ces structures de données à partir des possibilités de chaque langage.

Une implantation d'une structure de donnée repose sur :

- une représentation mémoire choisie,
- une implémentation algorithmique des opérations de la structure de donnée.
- 2 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.

Mais 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 ensembles (collection sans doublons)

- HashSet<E> : ensemble implanté sous forme de paires clef/valeur,
- TreeSet<E> : ensemble implanté sous forme d’arbre balancé.

- Structure 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

Etudier 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"

Un problème de fond des tableaux :

- ils sont créés au run-time.
- une fois créés : la taille ne peut pas être modifiée.

3.1. Exercice : comment "agrandir/rétrécir" un tableau ?

Question : Soit un tableau rempli de longueur n, dont la référence est stockée dans la variable `tabCourant` :

Comment faire "évoluer" la taille du tableau rempli, par exemple l’agrandir (tabCourant donnera accès à un tableau "plus grand").

Rappel : un tableau se manipule au travers de sa référence.

Réponse :

```
.
```

```
.
```

```
.
```

3.2. Exercice : le faire …

Dans le programme suivant, indiquer le code nécessaire à ajouter pour agrandir le tableau référencé par tab :

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

```
Scanner lect = new Scanner (System.in) ;
int tabCourant[], i, rep, nbValSuppl;
tabCourant = new int [5];
for (i=0;i< tabCourant.length; i++) {
    tabCourant[i] = lect.nextInt();  // ❶ Remplissage
}

System.out.println("   Nb Valeurs en plus ? : -> ");
nbValSuppl = lect.nextInt();
// A faire ici









for (i=5;i< tabCourant.length; i++) { // de 5 au nb de valeurs saisies
   tabCourant[i] = lect.nextInt();   // ❷ Remplissage complémentaire
}
for (i=0;i< tabCourant.length; i++) {
    System.out.println(tabCourant[i]);
    // Affichera les 5 valeurs saisies au départ ❶
    // + les valeurs ajoutées en ❷
}
```

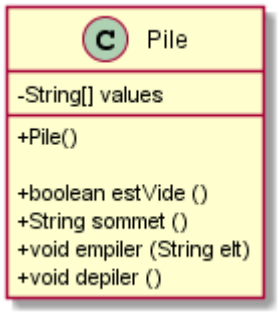
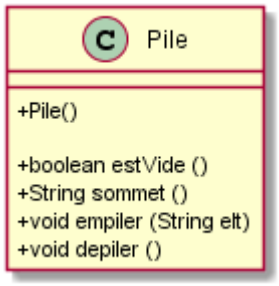
4. Classe à développer : Pile de String

La pile :

- Structure 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 (Le type Pile a déjà été implémenté sous forme d'un tableau de taille fixe, 100 éléments par exemple).

Diagramme UML complet 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 elt)` : permet d'ajouter un élément 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

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

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

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`.

Dernière mise à jour 2014-02-22 19:24:13 CET