INF2010 – Structures de données et algorithmes
Automne 2015
Travail Pratique 2
Structures de données séquentielles

Objectifs :

- Implémenter une file à partir d'un tableau et d'une liste chainée.
- Implanter un algorithme de résolution d'expressions postfixe à l'aide d'une pile
- Implanter un algorithme pour trier une pile

Problème 1: File (2.5 points)

Vous devez compléter l'implémentation d'une file. Pour rappel, une file est une structure FIFO (first-in-first-out). Un élément peut donc seulement être ajouté à la fin de la file et seul l'élément en tête de file peut être retiré.

Vous allez implémenter une file en utilisant les deux structures de données suivantes :

File à partir d'un tableau (1.25):

Un tableau est une séquence de cases auxquelles on accède par leur index et qui contiennent les données de la file à raison d'une donnée par case.

File à partir d'une liste chainée (1.25):

Une liste chaînée est une séquence de nœuds chaînés, c'est-à-dire d'objets distincts qui contiennent les données de la liste, à raison d'une donnée par noeud. Les implantations de listes chaînées requièrent donc généralement l'utilisation d'une classe interne `Node` contenant un champ destiné aux données et un autre destiné au chaînage.

Vous trouverez dans les fichiers `ArrayQueue.java` et `LinkedListQueue.java` la description des méthodes à compléter. La fonction `main` du fichier `QueueMain.java` vous permettra de tester vos deux classes.

Problème 2: Manipulation de piles (2.5 points)

Une pile est un conteneur qui implémente le protocole dernier entré, premier sorti (LIFO, "Lastin-First-out). C'est l'une des structures de données les plus utilisées en informatique, notamment pour la récursivité, le traitement des expressions arithmétiques et les parcours d'arbres et de graphes.

Exercice 1 (1.5):

Utilisez une pile `java.util.Stack` pour implanter un algorithme de résolution d'expression en notation postfixe. La notation postfixe s'écrit de manière à ce que chaque opérateur (pour cet exercice, les opérateurs permis sont : +, -, * et /) suivent immédiatement ses opérandes dans l'expression. Cette notation permet d'écrire des expressions arithmétiques sans jamais avoir recours aux parenthèses. Par exemple, l'expression infixe (1+2) * 4 s'écrit 1 2 + 4 * en notation postfixe (les espaces sont importants pour délimiter les nombres!). Implantez votre résolveur d'équation postfixe dans la classe `PostfixSolver`.

Exercice 2 (1):

Implémentez une fonction qui permet de trier les éléments d'une pile de `java.util.Stack` en ordre croisant. L'élément le plus petit doit donc se trouver au-dessus de la pile. Votre fonction peut utiliser une deuxième pile pour effectuer le tri, mais elle ne peut pas utiliser aucune autre structure de données. Implantez votre fonction dans la classe `SortStackMain`.
De plus, bien que `java.util.Stack` hérite des méthodes de `java.util.Vector`, vous n'êtes autorisés qu'à utiliser les méthodes spécifiques de la pile soient `peek()`, `pop()`, `push()` et `empty()`. Vous serez évalués quant à l'efficacité de votre implantation.

Instructions pour la remise

Le travail doit être fait par équipe de 2 personnes et doit être remis via Moodle au plus tard le 7 octobre pour le groupe 2 et le 14 octobre pour les groupes 1 et 3 avant 23h55.

Veuillez envoyer vos fichiers .java **seulement**, dans un **seul répertoire**, le tout dans une archive de type *.**zip** (et seulement **zip**, pas de **rar**, **7z**, etc) qui portera le nom : **inf2010_lab1_MatriculeX_MatriculeY.zip**, où MatriculeX < MatriculeY.

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard. Si votre dépôt ne respecte pas la nomenclature définie ci-dessus, 0.5 point de pénalité sera appliqué.