TP n°7

Piles et files

1 Les piles

Une pile est une structure de données qui fonctionne comme une pile d'assiettes, l'assiette qui est au-dessus étant le sommet de la pile. Avec une structure de données de type pile, les seules opérations possibles sont *empiler*, *dépiler*, et *test du vide*.

Lorsqu'on dépile, on commence par l'élément qui est au sommet de la pile (structure LIFO *Last In, First Out*).

On se restreint d'abord à des piles d'entiers dont la taille est bornée par `tailleMax`. Lorsque la pile est pleine on ne peut plus empiler et lorsque la pile est vide on ne peut pas dépiler ; en pratique on teste le vide avant de dépiler. On peut implanter ces piles avec la classe suivante :

```
public class Pile {
  private static final int tailleMax = 100; // taille maximale de la pile
  private int[] contenu; // éléments contenus dans la pile
  private int sommet; // position du sommet de la pile dans le tableau
}
```

L'élément `contenu[sommet]` est l'élément qui se trouve en haut de la pile. Par convention, la pile est vide si `sommet` vaut -1.

Exercice 1 Ecrire un constructeur `Pile()` qui initialise une pile vide.

Exercice 2 Ecrire les méthodes `empiler`, `depiler` et `estVide`.

2 Calculateur à pile

On veut écrire un petit calculateur capable d'effectuer les opérations +, -, *, / (division entière), % (modulo) sur les nombres entiers. On utilisera une méthode analogue à celle utilisée dans les calculatrices HP.

Pour effectuer $56 + 23$, on procède ainsi dans un terminal :

```
> 56
> 23
> +
> =
> 79
> q
```

Ainsi chaque nombre lu est placée dans la pile ; quand on lit un opérateur, on dépile le nombre d'opérandes nécessaires, on effectue l'opération et on remet le résultat sur la pile. Le signe `=` provoque l'affichage du nombre placée au sommet de la pile et le signe `q` permet de terminer le calcul en cours et de quitter le programme.

Exercice 3 Comment va-t-on utiliser la classe précédente ?

Exercice 4 Ecrire la fonction `void applique(Pile p, String op)` qui applique l'opération `op` à la pile `p`.

Exercice 5 Ecrire la méthode `main` du programme du calculateur. On pourra utiliser la méthode `int Deug.stringToInt(String s)` pour convertir une chaîne de caractères en entier.

Exercice 6 On veut afficher l'expression arithmétique correspondant au résultat affiché. Par exemple, si l'on entre successivement les éléments suivants :

```
56 23 + 8 7 * + =
```

le calculateur affichera le nombre 135 avec l'expression arithmétique correspondante :

$$((56 + 23) + (7 * 8)) = 135$$

Pour cela on utilisera une pile de chaînes de caractères en plus de la pile d'entiers. Dans la pile de chaînes de caractères on ne conservera pas les résultats des opérations, mais les expressions intermédiaires.

1. Créer une nouvelle classe `PileS` pour les piles de chaînes de caractères.
2. Ecrire la fonction `void appliqueS(PileS p, String op)` qui dépile deux éléments s_1 et s_2 de la pile `p` et empile la chaîne de caractère "(" + s_2 + `op` + s_1 + ")".
3. Réécrire la méthode `main` du programme du calculateur afin d'afficher l'expression arithmétique et le résultat.

3 Les files

Une file est une structure de données avec laquelle les seules opérations possibles sont *enfiler*, *défiler*, et *test du vide*. On commence par défiler le premier élément mis dans la file (structure FIFO *First In, First Out*).

Une première implémentation de ce type de structure peut se faire ainsi :

```
public class File{
  private int tailleMax; // taille maximale de la file
  private int[] contenu; // éléments contenus dans la file
  private int base; // position de la base de la file dans le tableau
  private int sommet; // position du sommet de la file dans le tableau
  private int nbElements; // nombre d'elements dans la file
}
```

Exercice 7 Implémenter la classe file d'entiers avec un constructeur `public File(int taille)` et les méthodes `enfiler`, `defiler`, `estVide`.

Exercice 8 Dans cet exercice on utilisera la notation suivante pour une file f contenant n éléments : $f = (f_1, \ldots, f_n)$. Le premier élément ajouté de cette file est l'élément f_1 et le dernier élément ajouté de cette file est l'élément f_n.

Ecrire la fonction `static File regroupe(File a, File b)` qui étant données les files $a = (a_1, \ldots, a_n)$ et $b = (b_1, \ldots, b_m)$ renvoie :
– la file $(a_1, b_1, a_2, b_2, \ldots, a_n, b_n, b_{n+1}, \ldots, b_m)$ si $n < m$
– la file $(a_1, b_1, a_2, b_2, \ldots, a_m, b_m, a_{m+1}, \ldots, a_n)$ sinon
On a le droit de changer les files a et b pour créer la nouvelle file.

Exercice 9 Ecrire la fonction `static File regroupeTriee(File a, File b)` qui étant données deux files `a` et `b` triées dans l'ordre croissant construit une file triée avec tous les éléments de `a` et de `b`.

4 Piles et Files à partir des Listes

Exercice 10 Réécrire la classe Pile en utilisant les listes chaînées. Doit-on alors changer la classe contenant le programme du calculateur ?

Exercice 11 Réécrire la classe File en utilisant les listes chaînées. On utilisera cette fois la classe `Liste` dans laquelle on a stocké le dernier élément dans l'attribut `fin`.