TP structures de données

Hugues Talbot

22 Avril 2005

1 Introduction

Dans ce TP nous étudions les structures de bases en utilisant des exemples simples.

2 Piles

Voici une mise en oeuvre d'une pile avec des tableaux.

```
    public class pArrayStackInt{
        protected int head[];
        protected int pointer;

        public pArrayStackInt(int capacity){
            head = new int[capacity];
            pointer = -1;
        }
        public boolean isEmpty(){
            return pointer == -1;
        }
        public void push(int i){
            if(pointer+1 < head.length)
                head[++pointer] = i;
        }
        public int pop(){
            if(isEmpty())
                return 0;
            return head[pointer--];
        }
    }
```

Et voici un programme de test:

```
import java.io.*;

class pArrayStackIntTest{
    public static void main(String[] args){
        pArrayStackInt s = new pArrayStackInt(10);
        int i,j;
        System.out.println("starting...");
        for(i=0;i<10;i++){
            j = (int)(Math.random() * 100);
            s.push(j);
            System.out.println("push: " + j);
        }
```

```
        while(!s.isEmpty()){
            System.out.println("pop: " + s.pop());
        }
        System.out.println("Fini.");
    }
}
```

Vérifiez que l'on obtient un résultat similaire à celui-ci:

```
starting...
push: 33
push: 66
push: 10
push: 94
push: 67
push: 79
push: 48
push: 7
push: 79
push: 32
pop: 32
pop: 79
pop: 7
pop: 48
pop: 79
pop: 67
pop: 94
pop: 10
pop: 66
pop: 33
Fini.
```

On vérifie que les premières valeurs entrées sont les dernières sorties.

Exercise 2.1 (Propriétés des piles)

1. *Bien comprendre comment la classe de pile fonctionne. Quelles sont les limites principales ?*
2. *Pour résoudre le problème du type de donnée, remplacer le type d'entiers par un type* `Object`.
3. *Modifiez le programme de test pour qu'il accepte un type d'objets quelconques.*
4. *Testez avec* `Integer` *et* `Float`.

5. *(plus dur) La pile telle qu'elle est présentée ici doit être initialisée avec une certaine taille. Comment pourrait-on s'affranchir d'une telle limite. Comment la mettre en oeuvre ?*

Maintenant nous allons tout faire avec des classes qui supportent des types d'objets arbitraires.

3 File d'attentes

Voici le squelette d'une classe de file d'attente.

```
public class pArrayQueue{
    protected Object[] array;
    protected int start,end;
    protected boolean full;

    public pArrayQueue(int maxsize){
        array = new Object[maxsize];
        start = end = 0;
        full = false;
    }

    public boolean isEmpty(){
        return ((start == end) && !full);
    }

    public void insert(Object o){
      // A completer
    }

    public Object remove(){
      // A completer
    }
}
```

et voici la classe de test:

```
import java.io.*;

class pArrayQueueTest{
    public static void main(String[] args){
        pArrayQueue q = new pArrayQueue(10);
        Integer j = null;
        int i;
        System.out.println("starting...");
        for(i=0;i<10;i++){
            j = new Integer((int)(Math.random() * 100));
            q.insert(j);
            System.out.println("insert: " + j);
        }
        while(!q.isEmpty()){
            System.out.println("remove: " + ((Integer)q.remove()));
        }
        System.out.println("Done ;-)");
    }
}
```

Exercise 3.1 (Propriétés des files d'attente)

1. *Completez la classe* `pArrayQueue`

2. *Verifiez votre mise en oeuvre avec la classe de test fournie.*

3. *Verifiez que la file d'attente fonctionne comme une LIFO.*

4 Listes

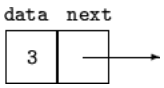
Dans cette section nous étudions la mise en oeuvre d'une liste simple.

Voici une mise en oeuvre possible pour un élément de liste:

4.1 ListNode

```
class ListNode {
    Object data;
    ListNode next;

    ListNode(Object data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

Un élément de liste a donc la forme suivante:



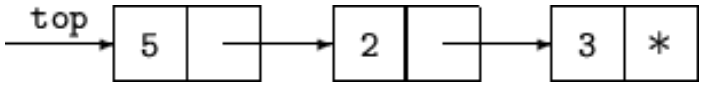
4.2 Piles et files d'attentes avec queues

En utilisant la classe du dessus, on peut mettre en oeuvre une classe de pile relativement simplement:

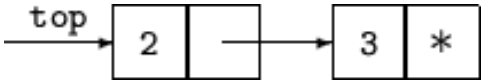
```
import java.util.NoSuchElementException;

public class pListStack  {

    private ListNode top;

    public pListStack() {
        top = null;
    }

    public boolean isEmpty() {
        return top == null;
    }

    public void push(Object item) {
        top = new ListNode(item, top);
    }

    public Object pop() {
```

```
        if (top == null) {
            throw new NoSuchElementException();
        } else {
            Object item = top.data;
            top = top.next;
            return item;
        }
    }

}
```

Quand la pile contient `[5, 2, 3]` le diagramme des objets est le suivant:



Lorsque 5 est enlevé, le diagramme devient le suivant:



Exercise 4.1 (Propriétés des listes)

1. *Mettez en oeuvre les classes* `ListNode` *et* `pListStack`.
2. *Modifiez le programme* `pArrayQueueTest` *pour tester cette dernière classe.*
3. *Y a t'il un avantage à utiliser une liste ?*
4. *Modifiez* `pListStack` *pour mettre en oeuvre* `pListQueue`. *Note: il peut être utile de conserver un pointeur vers l'arrière de la pile. Testez.*

4.3 Une liste chaînée complète

Pour mettre en oeuvre une liste chaînée complète, avec insertion, suppression et parcours aléatoire, il faut le concept d'*iterateur*.

5 Application des piles et des queues

Exercise 5.1 (Inversion de tableau) *Utilisez une pile pour inverser l'ordre d'un tableau.*

Exercise 5.2 (Josephus) *À l'aide d'une file d'attente, Résoudre le problème suivant (Josephus)*

> *In Herstein and Kaplansky's wonderful book Matters Mathematical (Chelsea Publishing, 1978), the following legend about the famous first-century historian Flavius Josephus is recounted:*

In the Jewish revolt against Rome, Josephus and 39 of his comrades were holding out against the Romans in a cave. With defeat imminent, they resolved that, like the rebels at Masada, they would rather die than be slaves to the Romans. They decided to arrange themselves in a circle. One man was designated as number one, and they proceeded clockwise killing every seventh man... Josephus (according to the story) was among other things an accomplished mathematician; so he instantly figured out where he ought to sit in order to be the last to go. But when the time came, instead of killing himself he joined the Roman side.

Pour simplifier le problème on complètera l'implémentation suivante:

```
public class Josephus {
  /** Solution of the Josephus problem using a queue. */
  public static Object Josephus(pListQueue Q, int k) {
        // a completer
  }
  /** Build a queue from an array of objects */
  public static pListQueue buildQueue(Object a[]) {
    pListQueue Q = new pListQueue();
    for (int i=0; i < a.length; i++)
      Q.insert(a[i]);
    return Q;
  }
  /** Tester method */
  public static void main(String[] args) {
    String[] a1 = {"Alice", "Bob", "Cindy", "Doug", "Ed", "Fred"};
    String[] a2 = {"Gene", "Hope", "Irene", "Jack", "Kim", "Lance", "Maryam", "Noemie"};
    String[] a3 = {"Mike", "Roberto"};
    System.out.println("First␣winner␣is␣" + Josephus(buildQueue(a1), 3));
    System.out.println("Second␣winner␣is␣" + Josephus(buildQueue(a2), 10));
    System.out.println("Third␣winner␣is␣" + Josephus(buildQueue(a3), 7));
  }
}
```

La sortie du programme devrait être la suivante:

```
> java Josephus
    Doug is out
    Bob is out
    Alice is out
    Cindy is out
    Fred is out
First winner is Ed
    Irene is out
    Maryam is out
    Kim is out
    Lance is out
    Hope is out
    Noemie is out
    Gene is out
Second winner is Jack
```

```
    Roberto is out
Third winner is Mike
```