ITI 1521. Introduction à l'informatique II

Pile : éléments chaînés

by

Marcel Turcotte

Version du 12 février 2020

Préambule

Préambule

Aperçu

Pile : éléments chaînés

Nous implémentons une pile à l'aide d'éléments chaînés.

Objectif général :

- Cette semaine, vous serez en mesure d'implémenter une pile à l'aide d'éléments chaînés.

Préambule

Objectifs d'apprentissage

Objectifs d'apprentissage

- **Implémenter** une pile à l'aide d'éléments chaînés.
- **Comparer** les implémentations à l'aide de tableaux et d'éléments chaînés d'une pile.

Lectures :

- Pages 75-83, 157-159 de E. Koffman et P. Wolfgang.

Préambule

Plan du module

Plan

Implémentation à l'aide d'éléments chaînés

Implémentation d'une pile à l'aide d'éléments chaînés

Éléments chaînés

Objectifs d'apprentissage :

- **Implémenter** une pile à l'aide d'éléments chaînés.
- **Comparer** les implémentations à l'aide de tableaux et d'éléments chaînés d'une pile.

Lectures :

- Pages 90–95, 167–169 de E. Koffman et P. Wolfgang.

Implémentation à l'aide d'éléments chaînés

Rappel

Retour sur l'implémentation d'une pile à l'aide d'un tableau

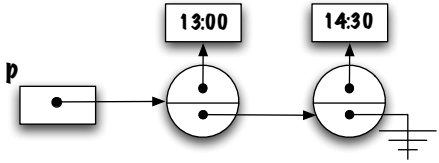
- L'**accès aux éléments** d'un tableau est **très rapide**, il nécessite toujours un nombre constant d'opérations.
- Cependant, puisque les tableaux ont une **taille fixe**, il y a certaines applications pour lesquelles ils ne sont pas appropriés.
- Une technique fréquemment utilisée afin de contourner cette limitation, consiste à copier les éléments du tableau dans un nouveau tableau, plus grand, et de remplacer l'ancien par le nouveau (**tableaux dynamiques**).
- Par contre, cela rend les insertions **plus coûteuses** (par rapport au temps d'exécution parce qu'il faut copier tous les éléments de l'ancien tableau vers le nouveau) et l'utilisation de mémoire est accrue parce que la taille **physique** de la structure de données sera généralement plus grande que sa taille **logique**.

Implémentation à l'aide d'éléments chaînés

Motivation

Structures chaînées

Considérons maintenant une implémentation utilisant toujours **une quantité de mémoire proportionnelle au nombre d'éléments** contenu dans la structure.



- Ces structures sont efficaces, au niveau du temps d'exécution (pour certaines opérations), parce qu'**elles évitent de recopier les éléments**.
- Les structures considérées ici sont **linéaires**, c.-à-d. chaque élément possède un prédécesseur et un successeur (sauf pour le premier et le dernier élément).
- Au contraire des structures de données à base de tableaux, **les éléments de ces structures ne sont pas contigus en mémoire.**

Implémentation à l'aide d'éléments chaînés

Expérimentation

La classe Elem

Étudiez la déclaration suivante * :

```
class Elem<E> {
   E value;
   Elem<E> next;
}
```

- Qu'y a-t-il de particulier avec la définition d'**Elem** ?
- La variable d'instance **next** est une référence vers un objet de la classe **Elem**.
- Est-ce valide ?
- Essayez par vous même !

  ```
  > javac Elem.java
  ```

- Oui, **c'est valide**, bien que cette définition semble circulaire.

*La question de la visibilité des variables sera abordée sous peu.

À quoi ça sert ?

- Déclarer une variable de type **Elem** :

```
Elem<Time> p;
```

- Créer un objet de la classe **Elem** :

```
new Elem<Time>();
```

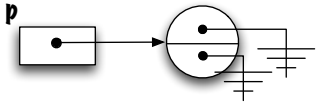
À quoi ça sert ?

- Sauvegarder la référence dans la variable **p**.

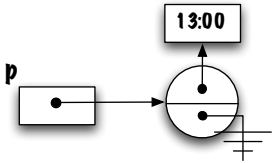
```
Elem<Time> p;
p = new Elem<Time>();
```

Notation : J'utiliserai toujours des cercles afin de représenter les objets de la classe **Elem**. La partie du haut représente la variable d'instance **value** alors que la partie du bas représente la variable **next**.

À quoi ça sert ?



- Comment change-t-on le contenu de la variable d'instance **value** de l'objet nouvellement créé ?

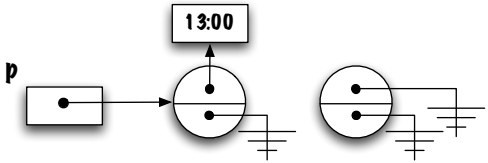


- On utilise la **notation pointée** (en anglais « *dot-notation* ») afin d'accéder aux champs de l'objet.

À quoi ça sert ?

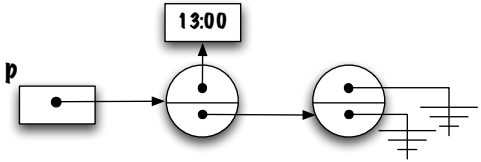
- Créer un nouvel objet de la classe **Elem**.

```
new Elem<Time>();
```



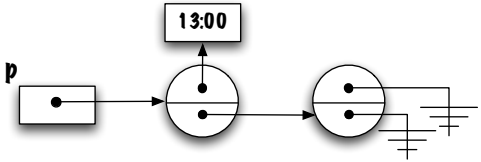
- Comment **chaîner** les éléments les uns aux autres ?

À quoi ça sert ?



- La variable **next** de l'objet désigné par la variable référence **p** reçoit la référence de l'objet **Elem** nouvellement créé.

À quoi ça sert ?

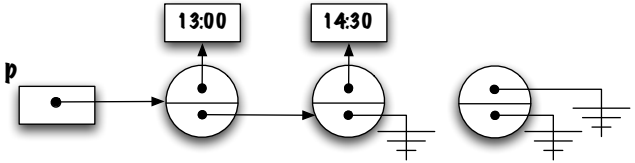


- Changer le contenu de la variable **value** de l'objet nouvellement créé.

À quoi ça sert ?

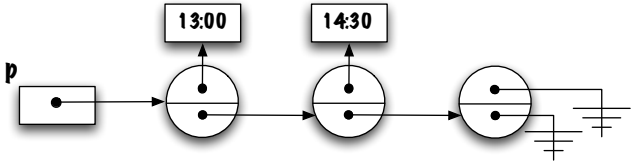
- Créer un nouvel objet de la classe **Elem**.

```
new Elem<Time>();
```



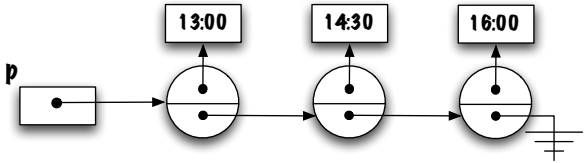
- Comment **chaîner** cet élément aux autres ?

À quoi ça sert ?



- Changer le contenu de la variable **value** de l'objet nouvellement créé.

À quoi ça sert ?




```
p.next.next = p;
```

- Que fait l'énoncé ci-haut ?

À quoi ça sert ?

```
p.next.next = p;
```

- Une structure **circulaire** a été créée !
- Le dernier élément **n'est plus accessible** ;
- Il sera récupéré par le ramasse-miettes ; **gc()**.

⇒ C'est la base des structures chaînées : des **informations** (valeurs) sont liées les unes aux autres par des **liens** (références).

Implémentation à l'aide d'éléments chaînés

Structures chaînées

Structures chaînées

```
class Elem<E> {
    E value;
    Elem<E> next;
}
```

Les structures de données chaînées, telles que celle-ci, nous permettent :

- de représenter des structures de données **linéaires**, telles que les piles, les files et les listes ;
- elles utilisent toujours une **quantité de mémoire proportionnelle au nombre d'éléments** ;
- tout ceci est rendu possible parce que la classe déclare une variable d'instance dont le type est une référence vers un objet de la même classe.

⇒ Lorsque les structures sont linéaires comme celles-ci, on parle alors de **listes (simplement) chaînées**.

Résumé

- Les **structures chaînées** sont une alternative aux **tableaux** pour sauvegarder des valeurs.
- Elles utilisent toujours une **quantité de mémoire proportionnelle au nombre d'éléments sauvegardés** puisque chaque élément est sauvegardé dans son contenant, un objet de la classe **Elem**. Chaque contenant est lié au suivant par une variable référence.
- Nous nous limitons aux **structures linéaires** pour l'instant, mais des structures de graphes ou des **arborescences** sont aussi possibles.

Implémentation à l'aide d'éléments chaînés

Constructeur

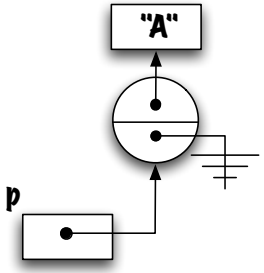
Constructeur

Voici le constructeur usuel de la classe **Elem** :

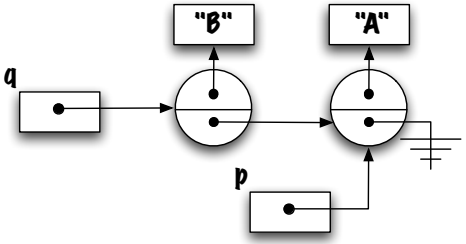
```
public class Elem<E> {

  E value;
  Elem<E> next;

  Elem(E value, Elem<E> next) {
      this.value = value;
      this.next = next;
  }
}
```

et son **usage habituel**,

```
p = new Elem<String>("A", null);
```




```
q = new Elem<String>("B", p);
```



Implémentation à l'aide d'éléments chaînés

Réaliser l'interface Stack

Implémenter une pile à l'aide d'éléments chaînés

```
public class LinkedStack<E> implements Stack<E> {

    public boolean empty() {

    }
    public void push(E o) {

    }
    public E peek() {

    }
    public E pop() {

    }
}
```

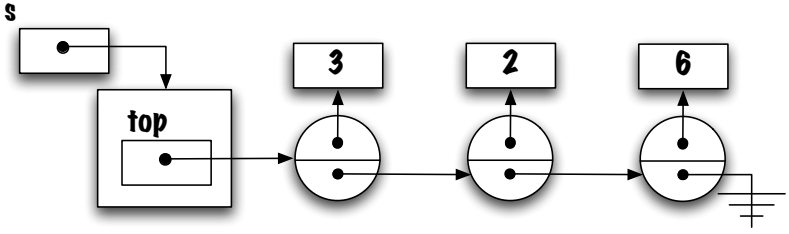
- Quelles sont les **variables d'instances** ?

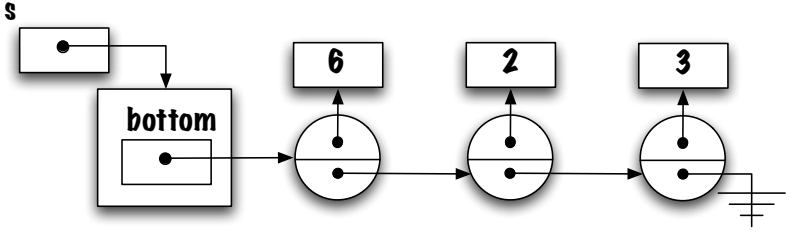
Implémentation à l'aide d'éléments chaînés

Variables d'instance

Quelles sont les variables d'instances ?

- Laquelle des deux stratégies suivantes est **préférable** ?





Discussion

Implémentation à l'aide d'éléments chaînés

Classe imbriquée interne

Classe Elem et principe d'encapsulation

La **visibilité des variables d'instance** n'est pas acceptable. C'est une violation du principe d'encapsulation.

- Quelles **solutions** s'offrent à nous ?

```java
public class Elem<E> {

  private E value;
  private Elem<E> next;

  public Elem(E value, Elem<E> next) {
      this.value = value;
      this.next = next;
  }
  public void setValue(E value) {
      this.value = value;
  }
  public void setNext(Elem<E> next) {
      this.next = next;
  }
  public E getValue() {
      return value;
  }
  public Elem<E> getNext() {
      return next;
  }
}
```

Java : classe imbriquée

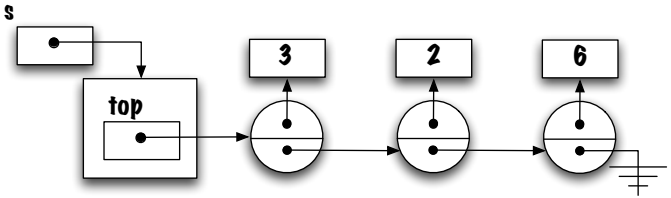
Java : classe imbriquée

- **Elem** est une classe **imbriquée** de la classe **LinkedStack**.
- Bien que la visibilité de la classe et de ses variables soit **private**, la classe **LinkedStack** a accès aux variables d'instance de la classe **Elem** parce que son implémentation est imbriquée.
- Pour l'instant, les classes imbriquées seront « static ». Nous les utiliserons comme si elles étaient des classes de premier niveau sauf que 1) la déclaration est imbriquée et 2) l'implémentation est accessible à la classe extérieure.
- Plus tard, nous verrons qu'il existe une seconde catégorie de classes imbriquées.

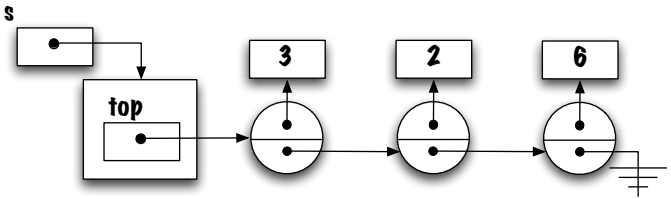
Implémentation à l'aide d'éléments chaînés

Implémentation des méthodes

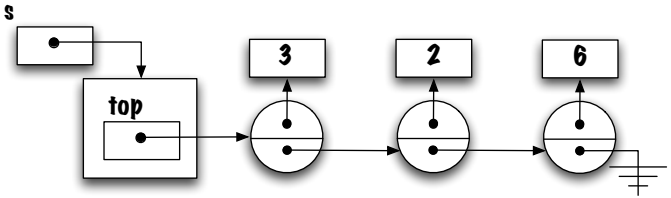
boolean isEmpty()



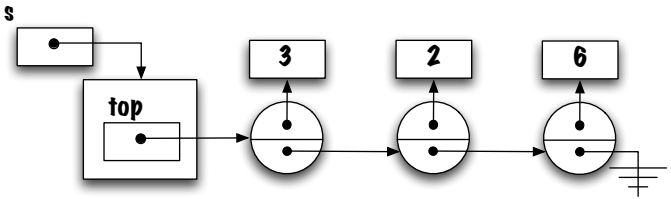
E peek()



void push(E value)



E pop()



Prologue

Résumé

- Le concept de **variable référence** est central pour les implémentations chaînées.
- La classe **Elem** possède deux variables d'instance, l'une d'elles sert à sauvegarder un élément d'information, l'autre sert d'arrimage pour le prochain élément de la liste.

Prochain module

- Traitement des erreurs en Java : **Exception**

References I


E. B. Koffman and Wolfgang P. A. T.
Data Structures : Abstraction and Design Using Java.
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

Université d'Ottawa