

Rappels:

4) PILES



Types abstraits de données (Abstract Data Type)

- Type de données
 - Un ensemble de valeurs
 - Un ensemble d'opérations
- Structure de données
 - Un ensemble de données
 - Un ensemble de relations sur ces données
 - Ces relations permettent d'organiser les données et d'y accéder

Abstraction

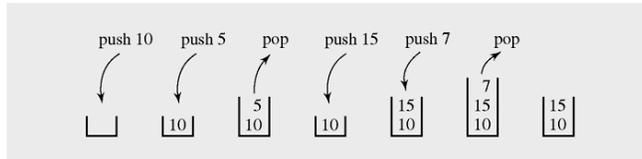
- Lorsqu'on utilise une structure de données, l'important est de connaître les opérations que l'on peut effectuer sur les données
 - ⇒ la façon dont les opérations sont programmées n'a pas d'intérêts pour l'utilisateur
- Un **type abstrait de données** est une description d'une structure de données comprenant:
 - type de données qu'on peut mettre en mémoire
 - description des opérations possibles sur ces données
 - conditions d'erreurs associées aux opérations

Type abstrait de données PILE (§4.2)

- Garde en mémoire des objets arbitraires
- Les insertions et suppressions se font dans l'ordre "dernier arrivé, premier sortie"
- Principales opérations:
 - `push(element)`: insère un objet sur la pile
 - `pop()`: retire et retourne l'objet en haut de la pile

Type abstrait de données PILE (suite)

FIGURE 4.1 A series of operations executed on a stack.



© 2005, Drozdek

Type abstrait de données PILE (suite)

opérations auxiliaires

- objet `haut()`: retourne l'objet en haut de la pile sans le retirer
- entier `taille()`: retourne le nombre d'objets de la pile
- booléen `estVide()`: indique si la pile est vide ou non

Implémentation de notre pile

- Interface JAVA correspondant à notre TAD pile

- On doit définir une classe `ExceptionPileVide`

- Différent de la classe JAVA intrinsèque `java.util.Stack`

```
public interface Pile {  
    public int taille();  
    public boolean estVide();  
    public Object haut()  
        throws ExceptionPileVide;  
    public void empiler(Object o);  
    public Object dépiler()  
        throws ExceptionPileVide;  
}
```

java.util.Stack

FIGURE 4.7 A list of methods in `java.util.Stack`; all methods from `Vector` are inherited.

Method	Operation
<code>boolean empty()</code>	Return <code>true</code> if the stack includes no element and <code>false</code> otherwise.
<code>Object peek()</code>	Return the top element on the stack; throw <code>EmptyStackException</code> for empty stack.
<code>Object pop()</code>	Remove the top element of the stack and return it; throw <code>EmptyStackException</code> for empty stack.
<code>Object push(Object e1)</code>	Insert <code>e1</code> at the top of the stack and return it.
<code>int search(Object e1)</code>	Return the position of <code>e1</code> on the stack (the first position is at the top; -1 in case of failure).
<code>Stack()</code>	Create an empty stack.

Utilisation de piles par la JVM

- La JVM garde en mémoire la chaîne des méthodes actives dans une pile
- Quand une méthode est appelée, la JVM empile un cadre contenant
 - les variables locales et les valeurs à retourner
 - un compteur qui pointe sur l'appel étant exécuté
- Quand une méthode est terminée, son cadre est retiré de la pile et le contrôle est donnée à la nouvelle méthode en haut de la pile

Première implémentation d'une pile

- Une façon simple d'implémenter un TDA pile est d'utiliser une liste



- On ajoute les éléments de gauche à droite
- Une variable t garde en mémoire le dessus de la pile

Algorithme *taille()*
retourner $t + 1$

Algorithme *dépiler()*
si *estVide()* alors
 throw *ExceptionListeVide*
sinon
 $t \leftarrow t - 1$
 retourner $P[t + 1]$

Première implémentation (suite)

- La liste gardant en mémoire les éléments de la pile peut devenir pleine



- L'opération "empiler" va alors envoyer une exception *ExceptionPilePleine*



Arrive seulement à cause de notre implémentation sous forme d'une liste

Algorithme *empiler(o)*
si $t = P.length - 1$ alors
 throw *ExceptionPilePleine*
sinon
 $t \leftarrow t + 1$
 $P[t] \leftarrow o$

Complexité et limitations

- Si N est la longueur de la liste utilisée dans l'implémentation
 - Complexité en espace: $O(N)$
 - Complexité en temps des opérations: $O(1)$
- Limitations
 - La longueur maximale de la liste doit être défini a priori et ne peut être changée
 - Essayer d'empiler un nouvel élément dans une liste pleine cause une exception (liée à l'implémentation)