Chapter 11: Récursivité

Presentation slides for

Java Software Solutions

Foundations of Program Design
Second Edition

by John Lewis and William Loftus

Java Software Solutions is published by Addison-Wesley

Récursivité

- La récursivité est une technique fondamentale de la programmation qui peut offrir une solution élégante pour certains problèmes
- Chapitre 11 se concentre sur:
  - Penser d'une manière récursive
  - Programmer d'une manière récursive
  - L'utilisation correcte de la récursivité
  - Exemples illustrant la récursivité

Pensée récursive

- Une définition récursive est une définition qui utilise le mot ou le concept en cours de définition dans la définition elle-même
- Lors de la définition d'un mot anglais ou français, souvent une définition récursive n'aide pas
- Mais dans d'autres situations, une définition récursive peut être une façon appropriée d'exprimer un concept
- Avant d'appliquer la récursivité à la programmation, il est mieux de se pratiquer à penser récursivement

Définitions Récursives

- Considérons la liste de nombres suivante:

```
24, 88, 40, 37
```

- Une telle liste peut être définie comme

```
Une LISTE est un:  nombre
          ou un:  nombre  virgule  une LISTE
```

- C'est-à-dire, une LISTE est définie comme étant un nombre ou un nombre suivi par une virgule suivi par une LISTE
- Le concept de LISTE est utilisé dans la définition de lui-même

Définitions récursives

- La partie récursive de la définition de LISTE est utilisée plusieurs fois et elle est terminée par une partie non-récursive:

```
nombre  virgule  LISTE
  24     ,       88, 40, 37

            nombre  virgule  LISTE

              88     ,       40, 37

                        nombre  virgule  LISTE

                          40     ,       37

                                    nombre
                                      37
```


Récursivité infinie

- Toutes les définitions récursives doivent avoir une partie non-récursive
- Sinon, il n'y aurait pas un moyen pour terminer le chemin récursif
- Une telle définition pourrait causer une récursivité infinie
- Ce problème est similaire à une boucle infinie, mais cette boucle fait partie de sa définition
- La partie non-récursive est souvent appelée la *base*

Définitions récursives

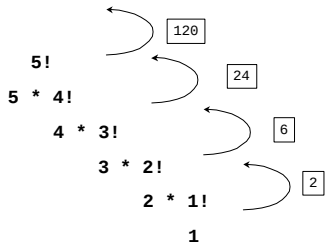
- N!, pour chaque entier positive N, est défini comme étant le produit de tous les entiers entre 1 et N inclusivement
- Cette définition peut être exprimée récursivement comme suit:

```
1!  =  1
N!  =  N * (N-1)!
```

- Le concept de factoriel est défini en terme d'un autre factoriel
- Finalement, la base de 1! est atteinte

7

Définitions récursives



8

Programmation récursive

- Une méthode en Java peut s'appeler elle-même. Si c'est le cas, elle est appelée méthode récursive
- Le code d'une méthode récursive doit être structuré de façon à clairement exposer le traitement de la base et le traitement de la partie récursive
- Chaque appel de la méthode initialise un nouvel environnement d'exécution avec de nouveaux paramètres et variables locales
- Comme toujours, quand une méthode termine son exécution, le contrôle retourne à la méthode qui l'a appelée (qui peut être la méthode elle-même)

9

Programmation récursive

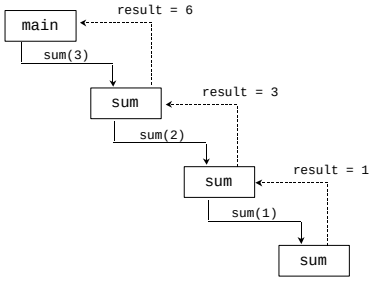
- Considérons le problème du calcul de la somme de tous les nombres entre 1 et tout entier positive N
- Ce problème peut être défini récursivement comme suit:

$$\sum_{i=1}^{N} = N + \sum_{i=1}^{N-1} = N + (N-1) + \sum_{i=1}^{N-2}$$

$$= \text{etc.}$$

10

Programmation récursive



11

Programmation récursive

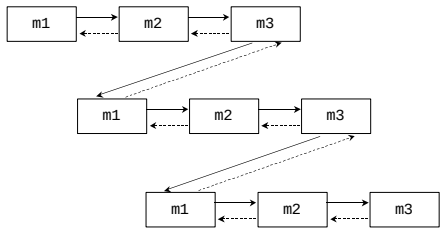
- Notez qu'utiliser la récursivité pour résoudre certains problèmes ne veut pas dire que l'on doit l'utiliser pour résoudre ces problèmes
- Par exemple, nous n'utilisons pas d'habitude la récursivité pour résoudre le problème de la somme des nombres entre 1 et N, parce que la version itérative est facile à comprendre
- Cependant, pour certains problèmes, la récursivité offre une solution élégante qui est souvent plus claire qu'une solution itérative
- Pour chaque problème, vous devez décider si la récursivité est la technique qui convient le mieux pour le résoudre

12

Récursivité indirecte

- Quand une méthode s'invoque elle-même, on a une *récursivité directe*
- Une méthode pourrait invoquer une autre méthode qui invoque une autre méthode, etc., jusqu'à ce que la méthode originale est encore une fois appelée
- Par exemple, la méthode `m1` pourrait invoquer `m2` qui invoque `m3` qui invoque encore une fois `m1`
- Ceci est appelé *récursivité indirecte*, et requiert une prudence comme pour le cas de la *récursivité directe*
- Elle est souvent plus difficile à tracer et à déboguer

Récursivité indirecte



Traverser un labyrinthe

- Nous pouvons utiliser la récursivité pour trouver un chemin dans un labyrinthe
- À partir de chaque location, nous pouvons chercher dans chaque direction
- La récursivité garde la trace du chemin à travers le labyrinthe
- La base est soit un mauvais mouvement ou lorsque la destination finale est atteinte
- Voir `MazeSearch.java` (page 472)
- Voir `Maze.java` (page 474)

Tours de Hanoi

- Les tours de Hanoi est un casse-tête fait à partir de trois piquets verticaux et quelques disques fixés sur ceux-ci
- Les disques sont de tailles variées. Ils sont initialement placés sur un des piquets. Le grand disque en bas et les autres, en ordre croissant des tailles, sont en haut
- L'objectif est de déplacer tous les disques d'un piquet à un autre tout en respectant les règles suivantes:
  - Nous ne pouvons déplacer qu'un seul disque à la fois
  - Nous ne pouvons pas déplacer un grand disque sur un petit disque

Tours de Hanoi

- Une solution itérative pour les tours de Hanoi est complexe
- Une solution récursive est courte et élégante
- Voir `SolveTowers.java` (page 479)
- Voir `TowersOfHanoi.java` (page 480)