

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

Presentation slides are copyright 2000 by John Lewis and William Loftus. All rights reserved.

Instructors using the textbook may use and modify these slides for pedagogical purposes.

Récurtivité

- La récurtivité 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 réursive
 - Programmer d'une manière réursive
 - L'utilisation correcte de la récurtivité
 - Exemples illustrant la récurtivité

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

Ω Considérons la liste de nombres suivante:

24, 88, 40, 37

Ω Une telle liste peut être définie comme

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

Ω C'est-à-dire, une LISTE est définie comme étant un nombre ou un nombre suivi par une virgule suivi par une LISTE

Ω 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
```

Récurtivité infinie

- ⌚ Toutes les définitions récursives doivent avoir une partie non-réursive
- ⌚ Sinon, il n'y aurait pas un moyen pour terminer le chemin récursif
- ⌚ Une telle définition pourrait causer une récurtivité infinie
- ⌚ Ce problème est similaire à une boucle infinie, mais cette boucle fait partie de sa définition
- ⌚ La partie non-réursive 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:

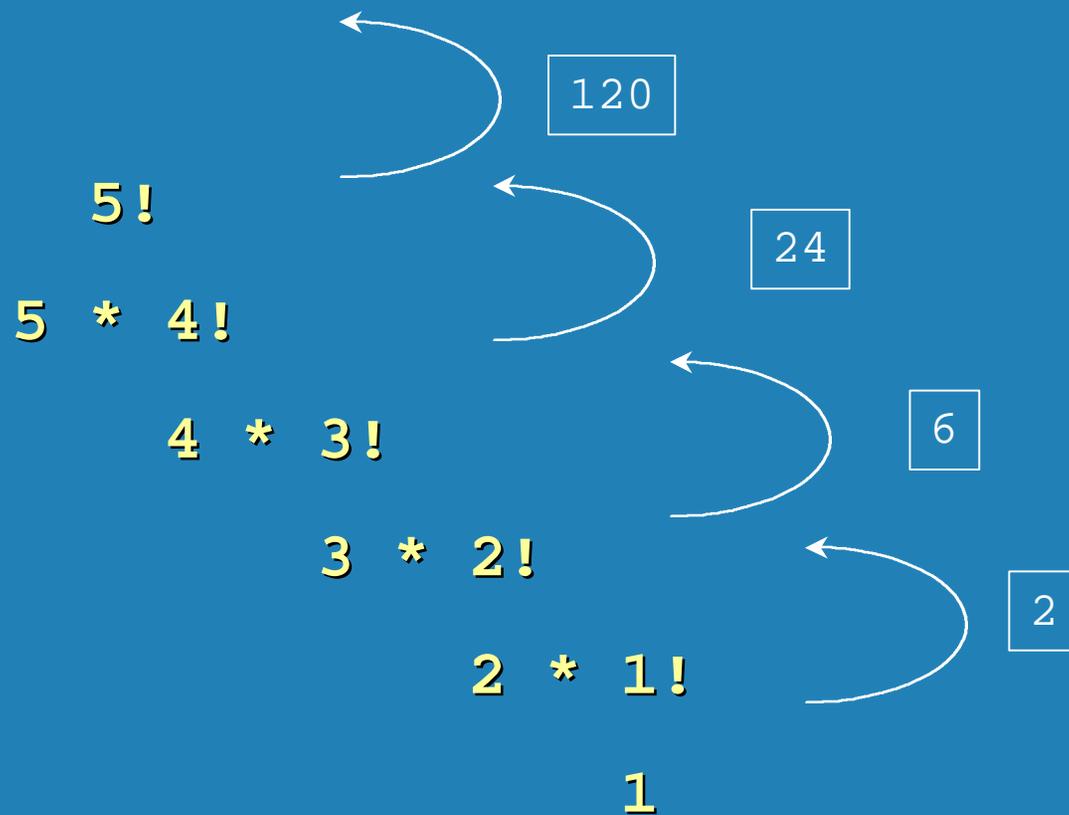
$$1! = 1$$

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

∞ Le concept de factoriel est défini en terme d'un autre factoriel

∞ Finalement, la base de 1! est atteinte

Définitions récursives



Programmation récursive

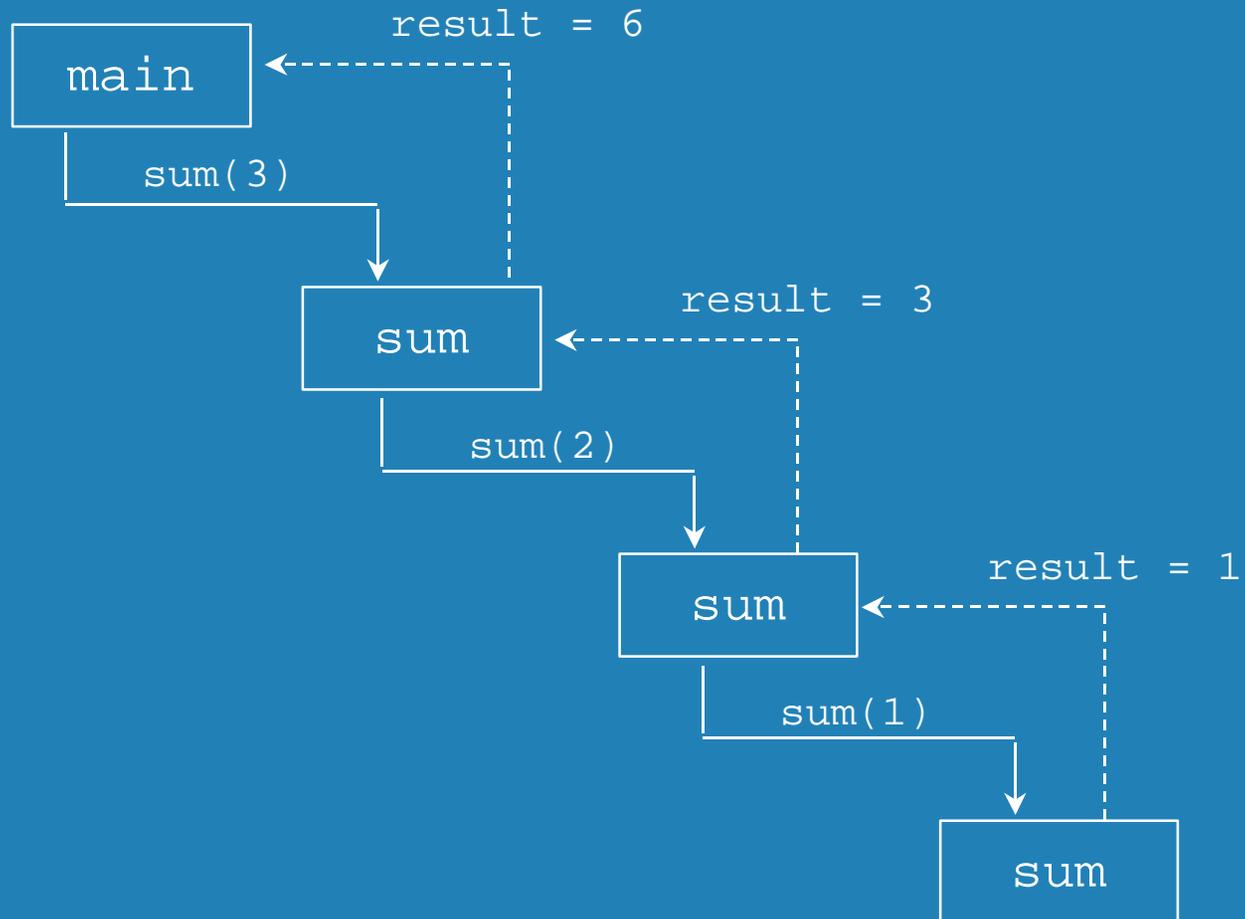
- ⌚ Une méthode en Java peut s'appeler elle-même. Si c'est le cas, elle est appelée méthode récursive
- ⌚ Le code d'une méthode récursive doit être structuré de façon à clairement exposer le traitement de la base et le traitement de la partie récursive
- ⌚ Chaque appel de la méthode initialise un nouvel 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)

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:**

$$\begin{aligned} \sum_{i=1}^N &= N + \sum_{i=1}^{N-1} = N + (N-1) + \sum_{i=1}^{N-2} \\ &= \text{etc.} \end{aligned}$$

Programmation réursive



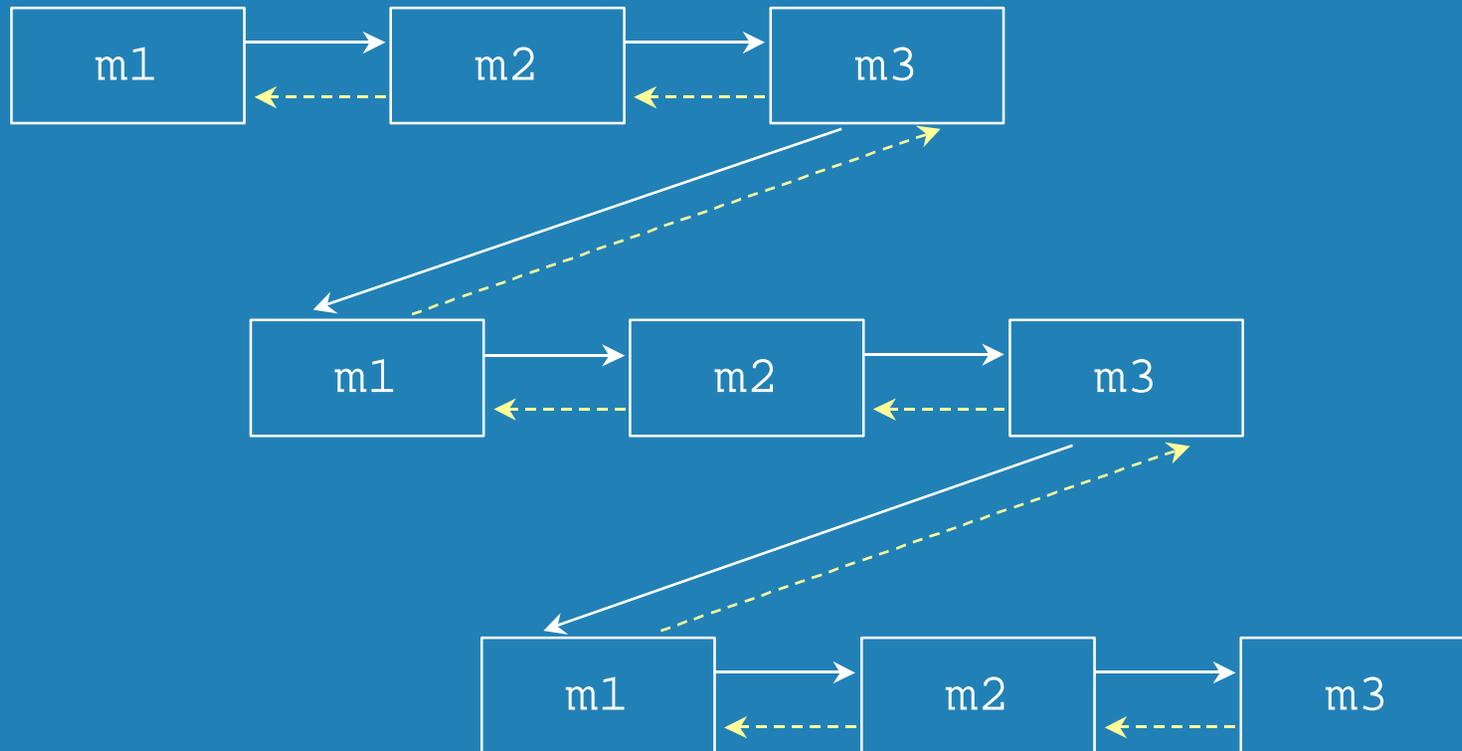
Programmation récursive

- ⌚ 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

Récurtivité indirecte

- ⌚ Quand une méthode s'invoque elle-même, on a une *récurtivité 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 *récurtivité indirecte*, et requiert une prudence comme pour le cas de la *récurtivité directe*
- ⌚ Elle est souvent plus difficile à tracer et à déboguer

Récurtivité 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)