TD 12,5 d'informatique UVSQ 2001-2002

Cours : Mathieu Le Coz – Marianne Durand
TD : Matthieu Finiasz

13 mars 2002

UN PETIT PEU DE RÉCURSION...

Introduction

Vous savez tous faire une boucle `for` : cela permet de créer des processus dits "itératifs". La boucle `while` est équivalente à une boucle `for` et permet elle aussi de faire des processus itératifs. Cependant, en algorithmique, il existe un concept beaucoup plus fort que celui de l'itération : la récursivité.

En java (ou dans tout autre langage de programmation), une méthode récursive sera une méthode qui s'appelle elle même avec d'autres paramètres. Cependant, pour que les appels successifs s'arrêtent un jour il faut en général préciser une condition d'arrêt (comme dans la boucle `for`). Voyons plutôt un exemple : on veut calculer la somme des entiers de 1 à `n`. Voila ce que ça donne :

```
static int Somme(int n){
   if (n == 0)                 <-- condition d'arret
      return 0;
   return (n + Somme(n-1));    <-- appel récursif
}
```

Ainsi un simple appel à `Somme(10);` va renvoyer directement 55. Dans le cas présent une boucle `for` aurait suffit pour faire le calcul, mais l'écriture récursive est compacte et élégante...

1 Un premier programme récursif

Dans cette question on va créer notre première méthode récursive. Pour cela commencez par créer un fichier `Recur.java` qui contient la classe `Recur`. Comme dans le TD du jeu de la vie cette classe n'a pas vraiment de sens propre : elle sert juste à contenir le `main` et les méthodes récursives qui seront appelées dans le `main`

On va chercher à calculer le factoriel d'un nombre. Pour l'instant le `main` va être vide et on ne le remplira qu'à la fin de la question. Créez donc une méthode `static long Fact(int n)` qui prend en argument un entier `n` dont on veut calculer le factoriel et qui, de manière récursive, calcule cette valeur. N'hésitez pas à vous inspirer de l'exemple au dessus pour écrire votre méthode.

Maintenant dans le `main` affichez (avec `System.out.println`) quelques exemples de valeurs retournées par `Fact`.

— **Compilez** et corrigez d'éventuelles erreurs —

Vous pouvez vérifier que pour 1 on obtient bien 1, pour 3 on obtient 6 et pour 9 par exemple on obtient 362880. ⚠ pour des valeurs trop grandes de `n` le résultat est faux car le résultat souhaité serait plus grand que la valeur maximale contenue dans un `long`.

2 Calcul de coefficients binomiaux

On va essayer de faire des calculs de coefficients binomiaux : C_p^n le nombre de manières de choisir p éléments dans un ensemble de n éléments. Afin de pouvoir vérifier les résultats obtenus on va faire le calcul de 2 façons différentes. Chacune des façon sera basée sur l'une des deux relations suivantes :

1. $C_p^n = C_{p-1}^{n-1} + C_p^{n-1}$
2. $C_p^n = \dfrac{n!}{(n-p)! \cdot p!} = \dfrac{n(n-1)\cdots(n-p+2)(n-p+1)}{p!}$

Commencez par écrire une méthode `static long Comb(int n, int p)` qui prend en arguments `n` et `p` et qui renvoie un `long` égal à C_p^n. Vous utiliserez pour cela la première relation. La méthode devrait être assez facile à écrire en s'inspirant de la méthode `Fact`, par contre il faut faire attention : la condition d'arrêt est un peu plus compliqué. Utilisez par exemple que quand $p = 0$, C_p^n vaut 1 quelque soit n et par ailleurs si $p = n$ on a aussi $C_p^n = 1$.

— **Compilez** et corrigez d'éventuelles erreurs —

Faites un appel à cette méthode dans le `main`, par exemple pour calculer C_3^{10} et vérifiez que vous obtenez bien 120.

On va maintenant programmer la deuxième méthode. Pour cela il suffit de créer un méthode `static long Prod(int n, int p)` qui calcul le produit $n(n-1)\cdots(n-p+2)(n-p+1)$. Là encore vous pourrez vous inspirer de la méthode `Fact`, il suffit de changer la condition d'arret.

Dans le `main` faites afficher le résultat de `Prod(n,p)/Fact(p)` pour quelques valeurs de `n` et `p` et comparez le résultat à celui obtenu avec la méthode `Comb`.

— **Compilez** et corrigez d'éventuelles erreurs —

Si vous le voulez, comme dans le jeu de la vie, vous pouvez maintenant faire lire des arguments passés sur la ligne de commande à votre programme. Comme cela, si vous tapez `java Recur 5 10` il vous calcule C_5^{10} des deux façons différentes et affiche les résultats.

3 Conversion d'une boucle `for`

Comme je l'ai écrit précédemment, la récursion est beaucoup plus forte que l'itération : on peut toujours réécrire de façon récursive une boucle `for`, alors que l'inverse n'est pas vrai. Certains problèmes ne peuvent être résolus qu'avec des alogrithmes récursifs.

Pour vérifier que l'on peut effectivement toujours convertir une boucle `for` en méthode récursive on va prendre d'anciens morceaux de code, et les réécrire avec une méthode récursive. De manière générale on peut convertir un code de la façon suivante :

```
static void Iter(arg1,arg2...){
   for (int i=0; i<n; i++){
      F(arg1,arg2,i...);
   }
}
```

```
static void Recu(i,arg1,arg2...){
   if (i<n){
      F(arg1,arg2,i...);
      Recu(i+1,arg1,arg2...);
   }
}
```

Où `F(arg1,arg2,i...)` représente tout ce qui est contenu dans le corps de la boucle `for`.

Recopiez les deux méthodes `randomize` et `afficher` du Td du jeu de la vie dans votre class Recur. En s'inspirant de l'exemple ci-dessus, essayez de réécrire la méthode `afficher` en utilisant une écriture récursive. Dans le `main` faites un essai d'affichage.

— **Compilez** et corrigez d'éventuelles erreurs —

Maintenant, essayez de réécrire la méthode `randomize`. ⚠ il faut retourner un `int[]` à la fin, donc c'est un peu plus compliqué.

— **Compilez** et corrigez d'éventuelles erreurs —