Chiffres romains — corrigé

Deug IF121 : Informatique Fondamentale

05 novembre 2003

1 Énoncé

Écrire un programme qui affiche un nombre entier positif en chiffres romains (cf. amphi 2).

On rappelle la valeur des chiffres romains :

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

On rappelle aussi les règles d'écriture des nombres en chiffres romains :

- *Normalement, on écrit les symboles du plus grand au plus petit (il peut y avoir plusieurs M, plusieurs C, plusieurs X, plusieurs I), et leur valeur s'ajoute. Par exemple, 172 s'écrit CLXXII.*
- *Si on se trouve avec quatre symboles identiques à la suite, on utilise à la place la notation soustractive : un symbole placé avant un symbole plus grand se retranche à lui. Par exemple, 199 s'écrit CXCIX.*

2 Démarche

Vu que la solution du problème n'est pas immédiate, on va essayer de le simplifier et de progresser par étapes.

Notons d'ores et déjà que nous ne chercherons qu'à traiter des nombres compris entre 1 et 3999 (les Romains ne connaissaient pas le zéro, et écrivaient rarement des nombres dépassant quelques milliers).

Une première simplification sera de commencer par la notation additive, qui ne tient compte que de la première règle (c'est-à-dire qu'on écrira, par exemple, VIIII pour 9). Une fois celle-ci mise au point, il sera temps de passer à la vraie écriture romaine en rajoutant la règle de soustraction.

3 Écriture simplifiée (additive)

3.1 Nombres de 1 à 9

En notation romaine additive, les nombres de 1 à 9 s'écrivent : I, II, III, IIII, V, VI, VII, VIII, VIII. On peut écrire facilement une méthode `unitésDécimales` qui prend un paramètre entier compris entre 1 et 9 et renvoie l'écriture romaine correspondante.

```
static String unitésDécimales (int n) {
  String s;
  switch (n) {
   case 1: s = "I"; break;
   case 2: s = "II"; break;
   case 3: s = "III"; break;
   case 4: s = "IIII"; break;
   case 5: s = "V"; break;
   case 6: s = "VI"; break;
   case 7: s = "VII"; break;
   case 8: s = "VIII"; break;
   case 9: s = "VIIII"; break;
   default: s = "?"; break;
  }
  return s;
}
```

(Java nous oblige à donner une valeur à la variable `s` dans tous les cas, même en dehors de l'intervalle qui nous intéresse, parce qu'il ne peut pas deviner que nous n'appellerons jamais la méthode avec un autre `n`.)

On peut raccourcir un peu le code en remarquant que les nombres 5 et suivants ressemblent aux précédents, mais avec `V` en tête. On construit maintenant `s` en deux phases.

```
static String unitésDécimales (int n) {
  String s = "";
  if (n >= 5) s = s + "V";
  switch (n % 5) {
   case 1: s = s + "I"; break;
   case 2: s = s + "II"; break;
   case 3: s = s + "III"; break;
   case 4: s = s + "IIII"; break;
  }
  return s;
}
```

Notons au passage que cette dernière méthode donne une sortie vide si `n` vaut 0.

3.2 Généralisation

En notation romaine simplifiée, on a successivement : de 0 à 3 fois M ; 0 ou 1 fois D ; de 0 à 4 fois C ; 0 ou 1 fois L ; de 0 à 4 fois X ; 0 ou 1 fois V ; de 0 à 4 fois I.

Mettons à part le cas du M. La méthode `unitésDécimales` nous donne les `V` et les `I` corrects, pour peu qu'on l'appelle non sur le nombre de départ N mais sur le chiffre des unités de N. Et en adaptant cette méthode pour qu'elle utilise les lettres `X` et `L`, ou `C` et `D`, au lieu de `I` et `V`, on peut obtenir tous les nombres de 1 à 999 (et même 0, avec une écriture vide). Voici donc une nouvelle méthode `unChiffreDécimal`, qui ressemble à `unitésDécimales` mais prend deux paramètres de plus qui indiquent quelles lettres utiliser.

```
static String unChiffreDécimal
                  (int x, char i, char v) {
  String s = "";
  if (x >= 5) s = s + v;
  switch (x % 5) {
   case 1: s = s + i; break;
   case 2: s = s + i + i; break;
   case 3: s = s + i + i + i; break;
   case 4: s = s + i + i + i + i; break;
  }
  return s;
}
```

3.3 Assemblage

Maintenant qu'on dispose de `unChiffreDécimal`, on peut écrire une méthode `convertitEnRomains` qui découpe un entier en morceaux (unités, dizaines, centaines, milliers), appelle `unChiffreDécimal` sur chaque morceau, et recolle le tout. *A priori*, `unChiffreDécimal` n'était pas prévu pour les milliers, mais comme on s'est limité à $\texttt{m} \leqslant 3$ elle donne aussi le bon résultat pour eux.

```
static String convertitEnRomains(int n) {
  int u = n % 10;
  int d = (n / 10) % 10;
  int c = (n / 100) % 10;
  int m = (n / 1000) % 1000;
  return (unChiffreDécimal(m, 'M', '?') +
          unChiffreDécimal(c, 'C', 'D') +
          unChiffreDécimal(d, 'X', 'L') +
          unChiffreDécimal(u, 'I', 'V'));
}
```

4 Écriture normale (additive et soustractive)

On reprend la même approche que précédemment : isolation d'une méthode `unChiffreDécimal` qui traite un chiffre de l'écriture du nombre en base 10. Cette fois, pour traiter 9 par une soustraction, il faut disposer d'une troisième lettre (`X` pour les unités, `C` pour les dizaines, `M` pour les centaines).

```
static String unChiffreDécimal
          (int n, char i, char v, char x) {
  String s;
  switch (n) {
   case 0: s = ""; break;
   case 1: s = "" + i; break;
   case 2: s = "" + i + i; break;
   case 3: s = "" + i + i + i; break;
   case 4: s = "" + i + v; break;
   case 5: s = "" + v; break;
   case 6: s = "" + v + i; break;
   case 7: s = "" + v + i + i; break;
   case 8: s = "" + v + i + i + i; break;
   case 9: s = "" + i + x; break;
   default: s = "???";
  }
  return s;
}
```

(Ici encore, il faut une valeur pour `s` dans le cas par défaut, même si en fait on appellera toujours la méthode avec `n` compris entre 0 et 9.)

La méthode `convertitEnRomains` est très similaire à l'écriture simplifiée, il faut juste passer un paramètre de plus à `unChiffreDécimal`.

```
static String convertitEnRomains(int n) {
  int u = n % 10;
  int d = (n / 10) % 10;
  int c = (n / 100) % 10;
  int m = (n / 1000) % 1000;
  return (unChiffreDécimal(m, 'M', '?', '?')
        + unChiffreDécimal(c, 'C', 'D', 'M')
        + unChiffreDécimal(d, 'X', 'L', 'C')
        + unChiffreDécimal(u, 'I', 'V', 'X'));
}
```

5 Programme complet

Voici un programme complet, qui fabrique à la fois l'écriture simplifiée et l'écriture normale. Les méthodes `additif` et `soustractif` sont les deux versions de `unChiffreDécimal` ; la méthode `convertitEnRomains` prend un paramètre supplémentaire qui indique si on souhaite ou non l'écriture simplifiée.

Le programme complet contient aussi une méthode `main` qui permet de tester la conversion.

Notons qu'en plusieurs endroits on a écrit des points d'interrogation (`'?'`) : ils n'apparaissent que dans des situations où ils ne sont en fait pas utilisés.

```
import fr.jussieu.script.Deug;
class ChiffresRomains {
    static String additif (int x, char un, char cinq) {
        String s = "";
        if (x >= 5) s = s + cinq;
        switch (x % 5) {
          case 1: s = s + un; break;
          case 2: s = s + un + un; break;
          case 3: s = s + un + un + un; break;
          case 4: s = s + un + un + un + un; break;
        }
        return s;
    }

    static String soustractif (int x, char un, char cinq, char dix) {
        String s;
        switch (x) {
          case 0: s = ""; break;
          case 1: s = "" + un; break;
          case 2: s = "" + un + un; break;
          case 3: s = "" + un + un + un; break;
          case 4: s = "" + un + cinq; break;
          case 5: s = "" + cinq; break;
          case 6: s = "" + cinq + un; break;
          case 7: s = "" + cinq + un + un; break;
          case 8: s = "" + cinq + un + un + un; break;
          case 9: s = "" + un + dix; break;
          default: s = "???";
        }
        return s;
    }

    static String convertitEnRomains(int n, boolean simplifié) {
        if (n >= 1 && n <= 4999) {
            int unités = n % 10;
            int dizaines = (n / 10) % 10;
            int centaines = (n / 100) % 10;
            int milliers = (n / 1000) % 1000;
            if (simplifié) {
                return (additif(milliers, 'M', '?') + additif(centaines, 'C', 'D') +
                        additif(dizaines, 'X', 'L') + additif(unités, 'I', 'V'));
            } else {
                return (soustractif(milliers, 'M', '?', '?') + soustractif(centaines, 'C', 'D', 'M') +
                        soustractif(dizaines, 'X', 'L', 'C') + soustractif(unités, 'I', 'V', 'X'));
            }
        } else return "Nombre impossible à écrire en chiffres romains.";
    }

    static void main (String[] args) {
        Deug.print("Entrez un nombre compris entre 1 et 3999 : ");
        int n = Deug.readInt();
        Deug.println("Notation romaine simplifiée : " + convertitEnRomains(n, true));
        Deug.println("Notation romaine vraie : " + convertitEnRomains(n, false));
    }
}
```