Récursivité

Plan

1. Définition et exemples
2. Récursivité croisée
3. Fractales

1. Définition et exemples

Un sous-programme est *récursif* s'il s'appelle lui-même en cours d'exécution.

C'est l'équivalent de la définition *par récurrence*.

Exemple 1 : factorielle

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{sinon.} \end{cases}$$

Exemple 2 : pgcd

$$\text{pgcd}(a, b) = \begin{cases} a & \text{si } b = 0 \\ \text{pgcd}(b, a \bmod b) & \text{sinon} \end{cases}$$

Exemple 3 : binomial $C(n, p)$

$$C(n, p) = \begin{cases} 1 & \text{si } p = 0 \text{ ou } p = n \\ C(n-1, p) + C(n-1, p-1) & \text{sinon} \end{cases}$$

Exemple 4 : fonction d'Ackermann

$$A(m, n) = \begin{cases} n + 1 & \text{si } m = 0 \\ A(m-1, 1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m, n-1)) & \text{sinon} \end{cases}$$

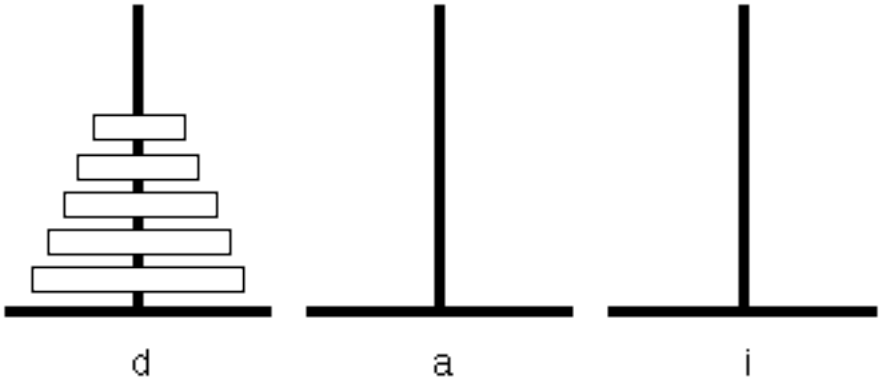
Exemple 5 : fonction "zéro"

$$z(n) = \begin{cases} n - 1 & \text{si } n > 0 \\ z(z(n+2)) & \text{sinon} \end{cases}$$

La programmation de fonctions définies par récurrence se fait de manière automatique.

```
static int Fact(int n)
{
  if (n == 0)
    return 1;
  else // facultatif ici
    return n * Fact(n-1);
}

static int C(int n, int p)
{
  if ((p == 0) || ( n == p))
    return 1;
  return C(n-1,p)+C(n-1,p-1);
}

static int pgcd(int a, int b)
{
  return (b != 0) ? a : pgcd(b, a % b);
}

static int zero(int x)
{
  if (x > 0) return x-1;
  return zero(zero(x+2));
}
```

Tours de Hanoi



Hanoi (n, d, i) :
  if $n > 0$ then
    Hanoi$(n-1, d, i)$;
    Déplacer(d, a);
    Hanoi$(n-1, i, a)$;

```
static void Deplacer (int d, int a) {
  System.out.println("Deplacer l'anneau de "+d+" sur "+a);
}
```

2. Récursivité croisée

C'est le cas lorsque plusieurs fonctions s'appellent mutuellement. Aucun problème en Java.

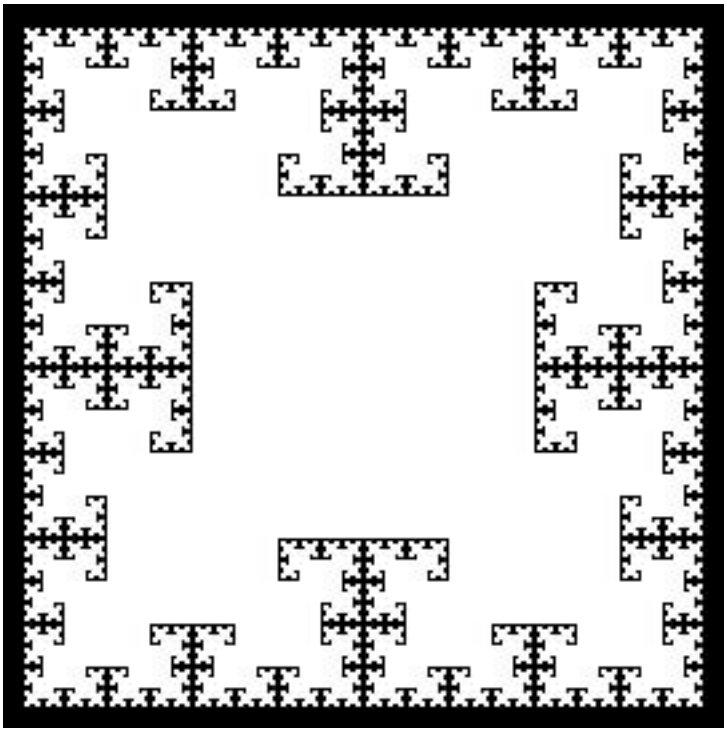
```
static boolean pair(int n) {
  return (n == 0) ? true : impair(n-1);
}

static boolean impair(int n) {
  return (n == 0) ? false : pair(n-1);
}
```

En Caml, la définition doit être simultanée:

```
let rec pair = function
    0 -> true
  | n -> impair(n-1)
and impair = function
    0 -> false
  | n -> pair(n-1);;
```

3. Fractales

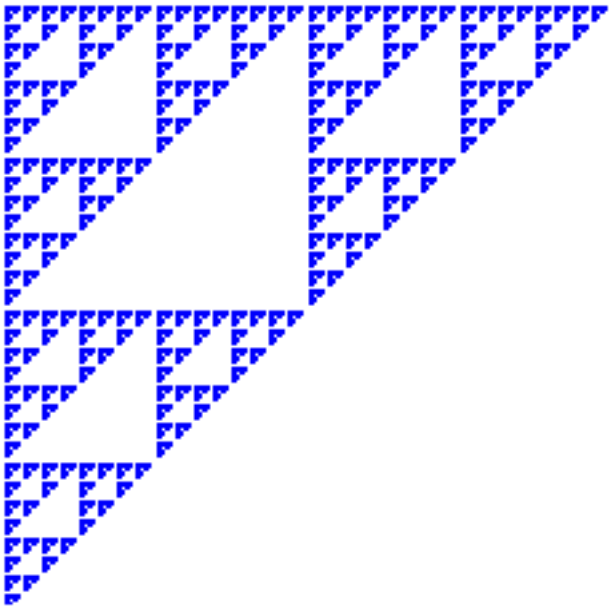
Les fractales autosimilaires se programment bien récursivement.

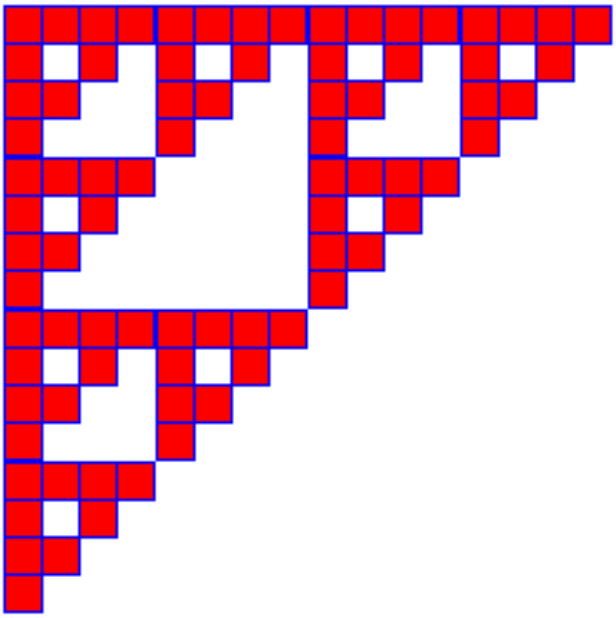


Cette fractale s'obtient en peignant des carrés blancs sur un fond noir.

```
void drawFractale(Graphics gr, int x, int y, int dim) {
  int cote = dim/2;
  if (cote < 1)
    return;
  int g = x - cote, h = y - cote;
  int d = x + cote, b = y + cote;

  drawFractale(gr, g, h, cote);
  drawFractale(gr, g, b, cote);
  drawFractale(gr, d, h, cote);
  drawFractale(gr, d, b, cote);

  gr.fillRect(g, h, d - g, b - h);
}
```

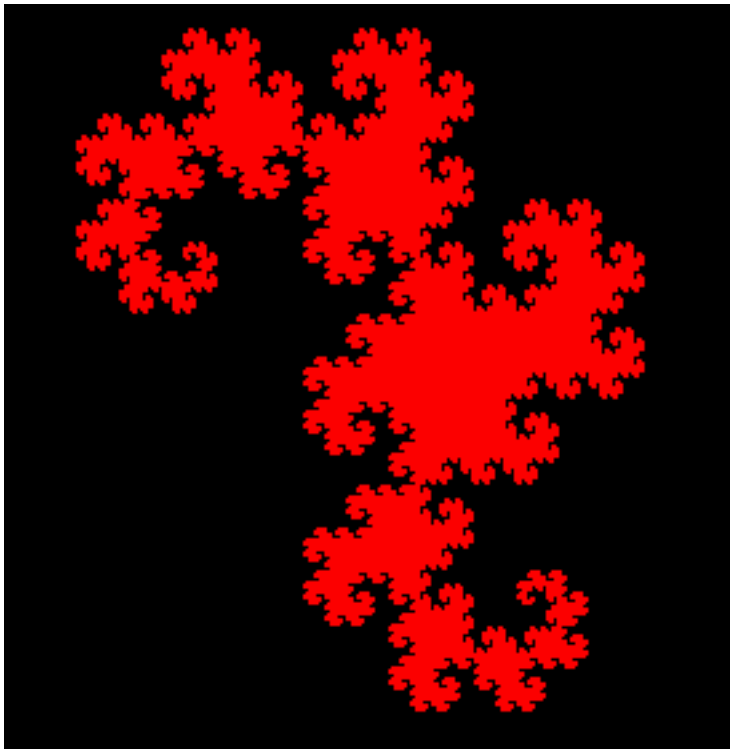
Le triangle de Sierpinski





```
void sierpinski(Graphics g, int x, int y, int dim) {
  if (n == 1)
    g.fillRect(x, y, dim, dim);
  else {
    int d = dim/2;
    sierpinski(g, n-1, x, y, d);
    sierpinski(g, n-1, x, y+d, d);
    sierpinski(g, n-1, x+d, y, d);
  }
}
```

La courbe du dragon



La fonction de tracé est

```
void dragon(Graphics g, int n,int x, int y,
  int z, int t) {
  if (n == 1)
    g.drawLine(x,y,z,t);
  else {
    int u = (x - y + z + t)/2;
    int v = (x + y - z + t)/2;
    dragon(g, n-1, x, y, u, v);
    dragon(g, n-1, z, t, u, v);
  }
}
```

Pour voir ces figures, le plus simple est d'utiliser une applette. Le cadre est

```
import java.applet.Applet;
import java.awt.*;

public class Dragon extends Applet {
  int n = 4;
  public void paint(Graphics g) {
    setBackground(Color.black);
    g.setColor(Color.red);
    dragon(g, n++, 100, 100, 228, 228);
  }
  void dragon(Graphics g, int n,int x, int y, int z, int t)
    {...}
}
```

Une applette est chargée par un navigateur, lors de la lecture d'une page qui le demande. Voici un bon prototype d'une telle page.

```
<APPLET CODE="Dragon.class" WIDTH=400 HEIGHT=400></APPLET>
```

Une indication de taille est nécessaire (par défaut, c'est zéro).