Séance 12: Récursivité et fractales

Université Paris Cité

Objectifs:

- Savoir écrire des fonctions récursives simples.
- Utiliser la récursivité pour dessiner des fractales.

Rappel : Le but de ce TP est de s'entraîner à programmer en utilisant la récursivité. Toutes les fonctions doivent donc être écrites de façon récursive, sans utiliser de boucles `for` ou `while` ! En cas de boucle infinie, vous verrez s'afficher la fameuse erreur `Stack Overflow`, par exemple :

```
Exception in thread "main" java.lang.StackOverflowError
  at Recursivite.factorial(Recursivite.java:4)
  at Recursivite.factorial(Recursivite.java:4)
  at Recursivite.factorial(Recursivite.java:4)
  at Recursivite.factorial(Recursivite.java:4)
  ......
```

Posez-vous alors les questions suivantes :
- Avez-vous pensé à inclure le cas de base de la récursion ?
- Est-ce que l'appel récursif se fait bien sur une entrée plus petite que celle de départ ?

1 Fonctions récursives basiques

Pour cette section, toutes les fonctions demandées sont à faire dans le fichier `Recursivite.java`. N'oubliez pas de tester vos fonctions après les avoir écrites !

Exercice 1 (Factorielle, ⋆)
Écrire une fonction récursive `int factorial (int n)` *qui calcule la factorielle d'un entier* $n \geq 0$.
Pour rappel, la factorielle est définie par $n! = 1 \times 2 \times \ldots \times n$, *avec la convention que* $0! = 1$. □

Exercice 2 (Fibonacci, ⋆)
Écrire une fonction récursive `int fibonacci (int n)` *qui calcule le* n*-ième terme de la suite de Fibonacci.*
Pour rappel, la suite de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ *et* $F_n = F_{n-1} + F_{n-2}$ *pour* $n \geq 2$. □

Exercice 3 (Syracuse, ⋆)
Le suite de Syracuse de premier terme $p \geq 1$ *est définie par :*

$$s_0 = p,$$
$$s_{n+1} = \begin{cases} s_n/2 & \text{si } s_n \text{ est pair,} \\ 3s_n + 1 & \text{si } s_n \text{ est impair.} \end{cases}$$

Écrire une fonction récursive `int syracuse (int p)` *qui renvoie le nombre d'étapes nécessaires pour que la suite de Syracuse de premier terme* p *atteigne 1.* □

Exercice 4 (Palindrome, ⋆)
On rappelle qu'un palindrome est une mot qui se lit de la même façon de gauche à droite et de droite à gauche. Écrire une fonction récursive `boolean palindrome (String s)` *qui renvoie* `true` *si la chaîne de caractères* `s` *est un palindrome, et* `false` *sinon.*

Indication : la fonction `s.substring(debut, fin)` *renvoie la sous-chaîne de* `s` *allant du caractère en position* `debut` *(inclus) jusqu'au caractère en position* `fin` *(exclus).* □

Exercice 5 (Somme de tableau, ⋆⋆)
Écrire une fonction `int sumArray (int[] tab)` *qui renvoie la somme des éléments du tableau* `tab`*.*

Indication : il sera utile de définir une fonction auxiliaire récursive `int sumArray (int[] tab, int i)` *qui renvoie la somme des éléments du tableau* `tab` *à partir de l'indice* `i`*.* □

Exercice 6 (Tableau trié, ⋆⋆)
Écrire une fonction `boolean isSorted (int[] tab)` *qui renvoie* `true` *si le tableau* `tab` *est trié dans l'ordre croissant, et* `false` *sinon.*

Indication : ici aussi, on commencera par écrire une fonction auxiliaire bien choisie ! □

2 Tortue et Fractales

On va à nouveau utiliser la « Tortue » graphique du TP 1b. Vous aurez donc besoin de télécharger les deux fichiers `Turtle.java` et `StdDraw.java` qui vous sont fournis. Les exercices sont à faire dans le fichier `Fractales.java`.

Pour faire ce TP, vous aurez besoin des fonctions suivantes :
- `turtle.left(angle)` : la tortue se tourne vers la gauche d'un angle donné (en degrés).
- `turtle.right(angle)` : la tortue se tourne vers la droite d'un angle donné (en degrés).
- `turtle.forward(distance)` : la tortue avance d'une distance donnée.
- `turtle.up()` : lève le crayon – la tortue se déplace sans dessiner de trait.
- `turtle.down()` : baisse le crayon – la tortue dessine un trait lorsqu'elle se déplace.

Au départ, la tortue est au centre de l'écran, tournée vers la droite, et le crayon est baissé.

Une fractale[1] est une figure géométrique auto-similaire, obtenue en répétant un motif de base à différentes échelles. Dans ce TP, nous allons dessiner quelques fractales célèbres en utilisant la récursivité.

2.1 Flocon de von Koch

Le flocon de von Koch[2] est une courbe fractale obtenue en partant d'un triangle équilatéral, et en remplaçant chaque segment de la figure par quatre segments plus petits, comme sur le dessin ci-dessous. Le flocon de von Koch d'ordre n est la figure obtenue en itérant ce processus n fois.

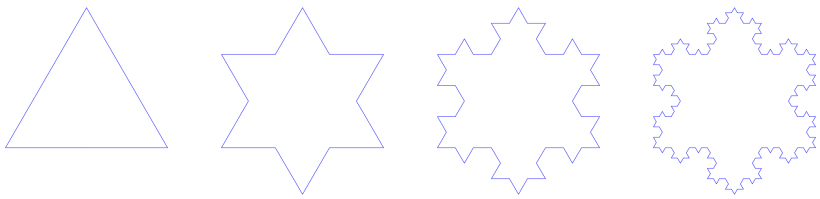
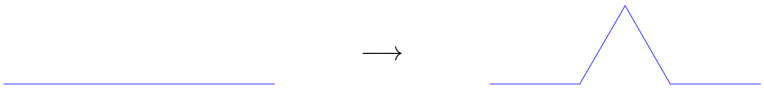


FIGURE 1 – Les quatre premières itérations du flocon de von Koch.

Dans un premier temps, nous allons dessiner une ligne de von Koch, qui est obtenue en itérant le même processus mais en partant d'un seul segment.

1. https://fr.wikipedia.org/wiki/Fractale
2. https://fr.wikipedia.org/wiki/Flocon_de_Koch



Plus précisément, chaque segment de longueur ℓ, est remplacé par un chemin composé de quatre segments de longueur $\ell/3$, construit de la manière suivante : avancer de $\ell/3$, tourner à gauche de 60 degrés, avancer de $\ell/3$, tourner à droite de 120 degrés, avancer de $\ell/3$, tourner à gauche de 60 degrés, avancer de $\ell/3$.

Par exemple, les lignes de von Koch d'ordre 2 et 3 sont dessinées ci-dessous :

Exercice 7 (Ligne de von Koch, $\star\star$)
Écrire une fonction récursive `void kochLine (int n, double length)` *qui dessine la ligne de von Koch d'ordre* `n`*, telle que la longueur entre les deux extrémités est égale à* `length`*.*

Indication : on remarque que la ligne de von Koch d'ordre n est formée de quatre lignes de von Koch d'ordre $n-1$. □

Exercice 8 (Flocon de von Koch, $\star$)
Écrire une fonction `void vonKoch (int n, double length)` *qui dessine le flocon de von Koch d'ordre* `n`*, obtenu à partir d'un triangle initial de longueur* `length`*.*

Indication : cette fonction n'est pas récursive, le flocon de von Koch d'ordre n est simplement formé de trois lignes de von Koch d'ordre n disposées en triangle. □

2.2 Triangle de Sierpiński

Le triangle de Sierpiński[3] est une figure fractale obtenue comme suit :
- Le triangle de Sierpiński d'ordre 0 est un triangle équilatéral.
- Le triangle de Sierpiński d'ordre $n+1$ est obtenu en dessinant trois triangles de Sierpiński d'ordre n plus petits, comme sur la figure ci-dessous.

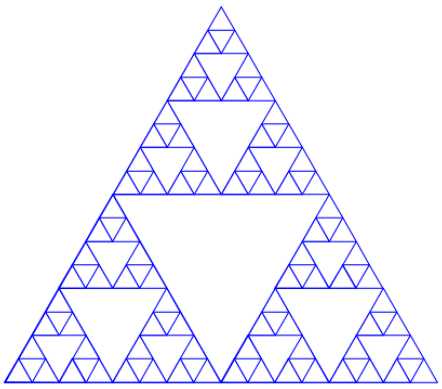


FIGURE 2 – Le triangle de Sierpiński d'ordre 4.

Exercice 9 (Triangle équilatéral, $\star$)
Écrire une fonction `void triangle (double length)` *qui dessine un triangle équilatéral de côté* `length`*. On fera attention à ce que la tortue revienne à sa position initiale et dans son orientation initiale à la fin de la fonction.* □

3. https://fr.wikipedia.org/wiki/Triangle_de_Sierpiński

Exercice 10 (Triangle de Sierpiński, ⋆⋆)
Écrire une fonction récursive `void sierpinski (int n, double length)` *qui dessine le triangle de Sierpiński d'ordre* `n` *et de côté* `length`. □

2.3 Courbe de Hilbert

La courbe de Hilbert[4] est une courbe fractale qui, lorsque le nombre d'itérations tend vers l'infini, passe de manière continue par tous les points d'un carré.

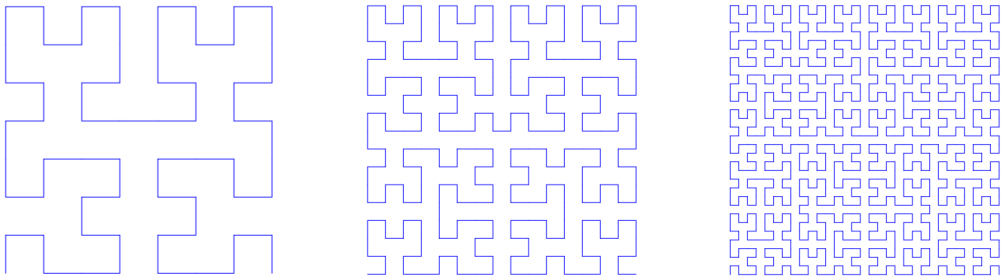


FIGURE 3 – Les courbes de Hilbert d'ordre 3, 4 et 5.

Pour construire cette courbe, nous allons d'abord générer un *mot de Hilbert*. Le mot de Hilbert d'ordre n est une chaîne de caractères composée des lettres `'A'`, `'B'`, `'F'`, `'-'`, `'+'`, obtenu en itérant le processus suivant :

- Le mot de Hilbert d'ordre 0 est simplement la chaîne `"A"`.
- Le mot de Hilbert d'ordre $n+1$ est obtenu à partir du mot de Hilbert d'ordre n, en remplaçant chaque occurrence de `'A'` par le mot `"-BF+AFA+FB-"`, et chaque `'B'` par le mot `"+AF-BFB-FA+"`.

Ainsi, les trois premiers mots de Hilbert sont :

- H_0 = `"A"`,
- H_1 = `"-BF+AFA+FB-"`,
- H_2 = `"-+AF-BFB-FA+F+-BF+AFA+FB-F-BF+AFA+FB-+F+AF-BFB-FA+-"`.

Exercice 11 (Mot de Hilbert, ⋆⋆)
Écrire une fonction récursive `String hilbertWord (int n)` *qui renvoie le mot de Hilbert d'ordre* `n`. □

Une fois le mot de Hilbert généré, on peut dessiner la courbe de Hilbert en interprétant chaque caractère du mot comme une instruction pour la tortue :

- `'F'` : avancer.
- `'-'` : tourner à gauche de 90 degrés.
- `'+'` : tourner à droite de 90 degrés.
- `'A'` et `'B'` : ne rien faire.

Exercice 12 (Courbe de Hilbert, ⋆⋆)
Écrire une fonction `void hilbert (int n, double length)` *qui dessine la courbe de Hilbert d'ordre* `n`*, qui remplit un carré de côté* `length`.

Pour cela, on génère le mot de Hilbert d'ordre n, puis on interprète chaque caractère du mot comme indiqué ci-dessus. À l'ordre n, chaque instruction `'F'` *doit avancer d'un segment de longueur* $\frac{\texttt{length}}{2^n}$. □

4. https://fr.wikipedia.org/wiki/Courbe_de_Hilbert