IUT de Montpellier **TD2 : diviser pour régner et fractales**
R3.02 Développement efficace

1 Exercices diviser pour régner

Exercice 1. Exponentiation rapide

Question 1.1.
Nous avons déjà écrit un algorithme récursif naïf `int puiss(int x, int n)` qui pour tout $n \geq 0$ et tout entier x calcule x^n en faisant $n-1$ multilpications. Nous allons maintenant appliquer le principe de diviser pour régner pour calculer plus rapidement x^n : cela s'appelle ici l'exponentiation rapide. L'idée est de constater que $x^n = x^{\frac{n}{2}} x^{\frac{n}{2}}$. En vous inspirant de cette remarque, écrire une algorithme `int puissRapide(int x, int n)` qui pour tout $n \geq 0$ et tout entier x calcule x^n.

Question 1.2.
Combien de multiplications fait votre algorithme ?

Exercice 2. Quicksort On considère un tableau d'entier t (que l'on souhaite trier par ordre croissant). Soit $p = t[t.length-1]$ que nous appellerons le pivot. L'idée du quicksort est la suivante. On va d'abord séparer t en deux selon la valeur p : pour un certain c on placera les valeurs plus petites ou égales à p dans les cases 0 à c (avec la valeur p dans la case c), et les valeurs strictement plus grandes que p dans les cases $c+1$ à $t.length-1$. On obtient donc deux sous tableaux plus petits ($t[0..(c-1)]$ et $t[(c+1)..(t.length-1)]$), et il n'y a plus qu'à les trier récursivement!

Question 2.1.
Ecrire un algorithme (récursif ou non, au choix) `int pivot(int []t, int i, int j)` qui pour tout t non vide, et tout i et j tels que $0 \leq i \leq j < t.length$, retourne une valeur c et réorganise les éléments de $t[i..j]$ en mettant (p dénote $t[j]$)

- les valeurs plus petites ou égales à p dans les cases i à c, avec la valeur p dans la case c
- et les valeurs strictement plus grandes que p dans les cases $c+1$ à j.

Question 2.2.
Ecrire un algorithme récursif `void quickSortAux(int []t,int i, int j)` qui pour tout tableau t non vide, pour tout i et j tels que $0 \leq i$ et $j < t.length$ (on peut donc avoir $i \geq j$) trie le sous tableau $t[i..j]$ selon le principe du quicksort.

Question 2.3.
En déduire un algorithme `void quickSort(int []t)` qui trie t selon le principe du quicksort.

2 Dessins de fractales avec Bob la tortue

Pour cette partie, commencez par récupérer les deux fichiers `DessinFractale.java` et `Turtle.java` dans le dossier `TP2` de la page gitlab du module. Vous devez placer ces deux fichiers dans le même dossier. Toutes les méthodes seront à écrire dans `DessinFractale.java`, et vous n'aurez même pas à ouvrir `Turtle.java`.

Question 2.4.
Vérifiez que vous pouvez bien compiler et exécuter `DessinFractale` : l'exécution doit ouvrir une fenêtre graphique avec la tortue dessinée, qui reste immobile pour le moment.

Le but de cette partie est de dessiner des fractales dans le plan. On considérera que le plan est orienté de façon usuelle comme indiqué sur la Figure 1. Pour effectuer des tracés dans le plan, on dispose de Bob : une tortue munie d'un crayon scotché sous son ventre, la pointe du crayon vers le sol. A tout moment, Bob est caractérisée par son abscisse, son ordonnée, sa direction (c'est à dire l'angle dans le sens trigonométrique entre entre la direction de l'axe des abscisses (de 0 vers ∞)) et la direction dans laquelle elle regarde, et la position du crayon (le crayon peut toucher le sol ou être relevé). On peut manipuler Bob grâce aux méthodes suivantes :

- `Turtle bob = new Turtle();` //créer Bob en position $(0,0)$, avec l'angle 0 (Bob regarde donc vers la droite), et le crayon qui touche le sol (et ouvre une fenêtre graphique)
- `bob.forward(double d)` //fait avancer Bob vers là où elle regarde d'une distance d

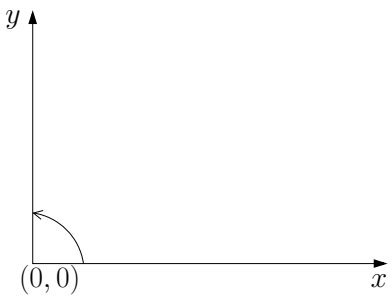

Figure 1: Orientation du plan

- `bob.left(double a)` //fait tourner Bob sur elle-même de a degrés vers la gauche
- `bob.right(double a)` //fait tourner Bob sur elle-même de a degrés vers la droite
- `bob.down()` //fait descendre le crayon pour qu'il touche le sol
- `bob.up()` //fait remonter le crayon pour qu'il ne touche plus le sol
- `bob.clear()` //efface l'écran
- `bob.speed(int v)` //modifie la vitesse de Bob (attention, plus v est grand, plus bob est lente)

Dans tout ce TP, les spécifications sont volontairement incomplètes : à vous de les compléter pour éviter les mauvaises surprises (en particulier, définir un prérequis sur la direction de Bob avant de commencer le tracé, et une spécification sur sa direction après le tracé). Pour tous les exercices, il est interdit d'écrire une méthode auxiliaire.

Exercice 3. Echauffement

Question 3.1.
Nous allons commencer par simplement dessiner un carré, sans rapport avec la récursivité ou les fractales, juste pour prendre en main Bob. Complétez la méthode `public void carre(double l)` pour qu'elle dessine un carré de côté l (cette spécification est volontairement incomplète, décidez vous même où le carré doit être tracé!).

Exercice 4. Flocon

Question 4.1.
Ecrire une méthode `public void vonKoch(double l, int n)` qui trace un flocon de Von Koch de taille l et d'ordre $n \geq 0$ (voir Figure 2).

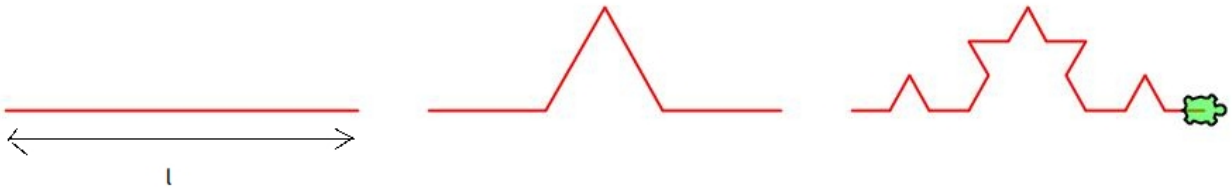

Figure 2: Exemples de flocons de Von Koch de taille l et d'ordre 0 (à gauche), 1 (au milieu) et 2 (à droite).

Exercice 5. Arbre

Question 5.1.
Ecrire une méthode pour tracer des arbres semblables à ceux dessinés Figure 3. Essayez de varier l'angle des branches!

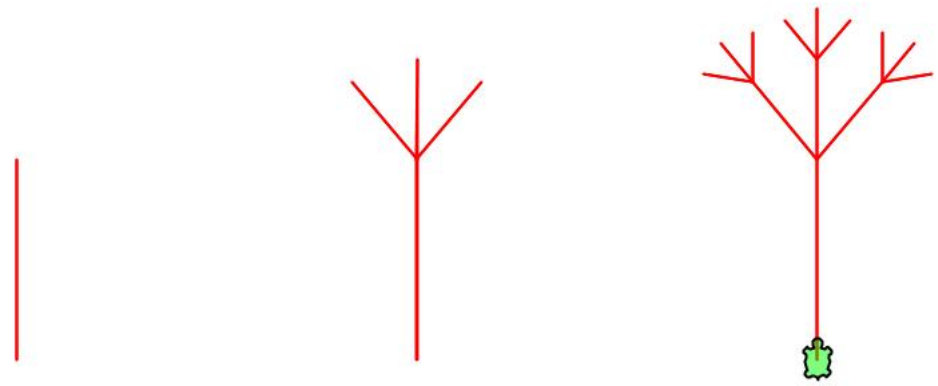

Figure 3: Exemples d'arbre d'ordre 0 (à gauche), 1 (au milieu) et 2 (à droite).

Exercices bonus

Exercice 6. Triforce (ou presque) (Sierpinski)

Question 6.1.
Ecrire une méthode pour tracer des figures semblables à celles dessinées Figure 4 (la taille totale du triangle englobant ne varie pas).

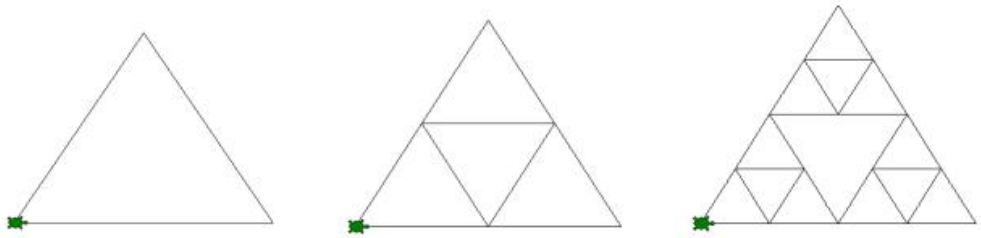

Figure 4: Exemples de Sierpinski à l'ordre 0, 1 et 2.

Exercice 7. Dragon

Question 7.1.
Ecrire une méthode pour tracer des dragons semblables à ceux dessinés Figure 5 (la taille des tracés n'est pas importante ici). Vous pouvez utiliser des méthodes auxiliaires, ou ayant des paramètres supplémentaires.

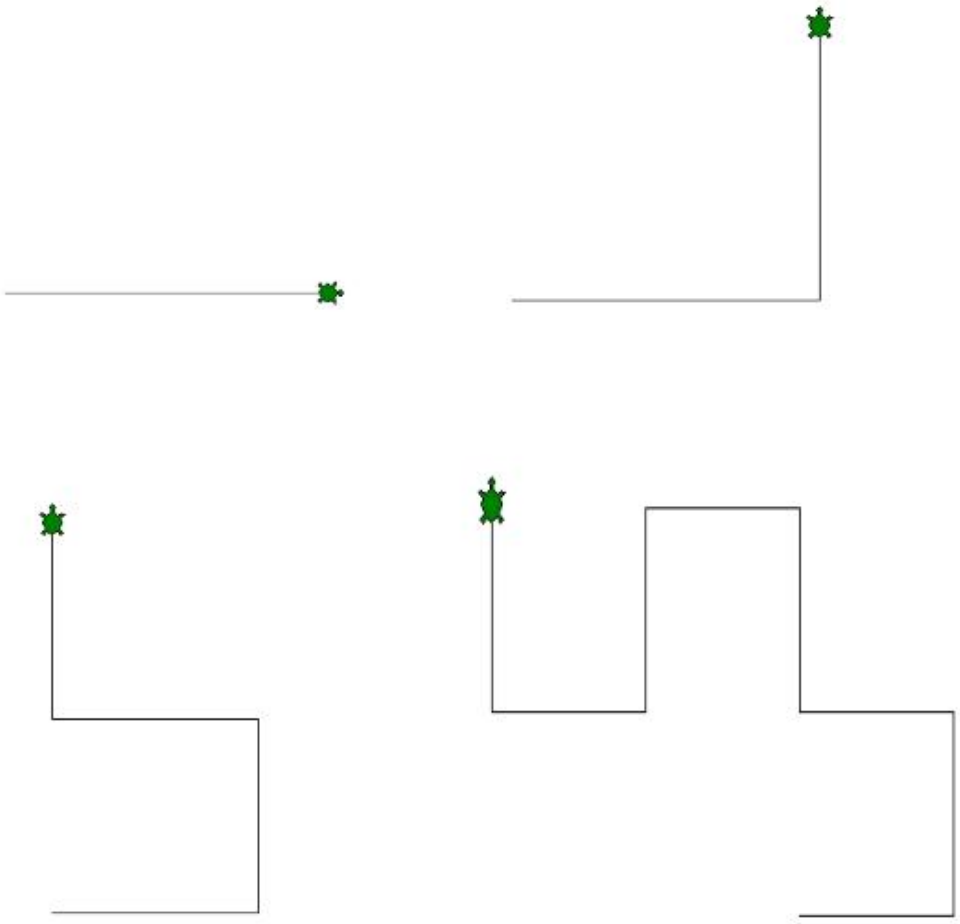

Figure 5: Exemples de dragons d'ordre 0, 1, 2 et 3.