Dessins de fractales avec Bob la tortue

Le but de ce TP est de dessiner des fractales dans le plan. On considérera que le plan est orienté de façon usuelle comme indiqué sur la Figure 1.

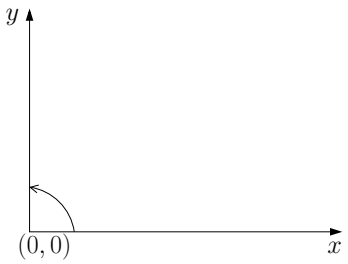

FIGURE 1 – Orientation du plan

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

— `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 1. Echauffement

Question 1.1.

Nous allons commencer par simplement dessiner un carré. Placez dans un même dossier les fichiers `Turtle.java` et `DessinFractale.java` disponibles sur l'ent. Tout le travail de ce TP se fera dans `DessinFractale.java`. 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 2. Diagonale

Question 2.1.
Ecrire une méthode `public void diago(double l, int n)` qui trace une diagonale de taille l et d'ordre $n \geq 0$ (voir Figure 2). Remarquez l'apparent paradoxe : le tracé "converge" vers une diagonale d'un carré de côté l, mais la longueur totale du tracé reste égale à $2l$ (et donc ne converge pas vers $\sqrt{2}l$).

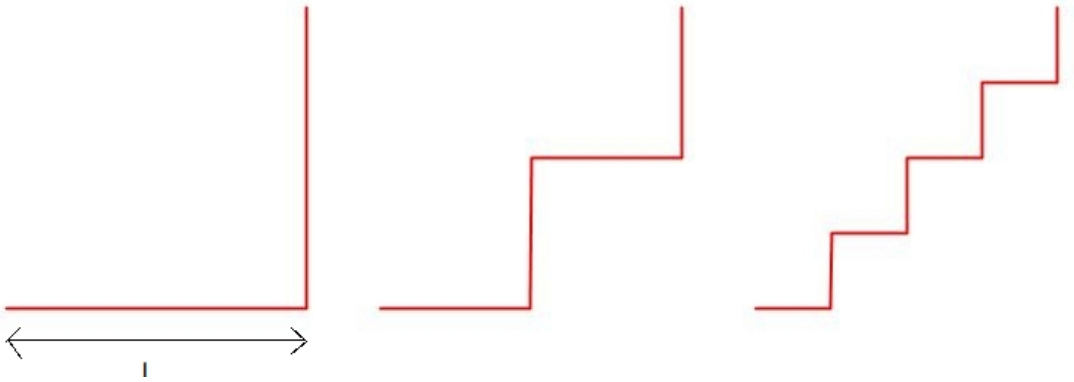

FIGURE 2 – Exemples de diagonales de taille l et d'ordre 0 (à gauche), 1 (au milieu) et 2 (à droite).

Exercice 3. Flocon

Question 3.1.
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 3).

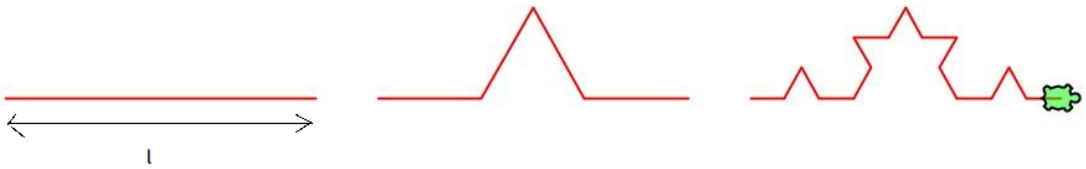

FIGURE 3 – Exemples de flocons de Von Koch de taille l et d'ordre 0 (à gauche), 1 (au milieu) et 2 (à droite).

Exercice 4. Arbre

Question 4.1.

Ecrire une méthode pour tracer des arbres semblables à ceux dessinés Figure 7. Essayez de varier l'angle des branches !

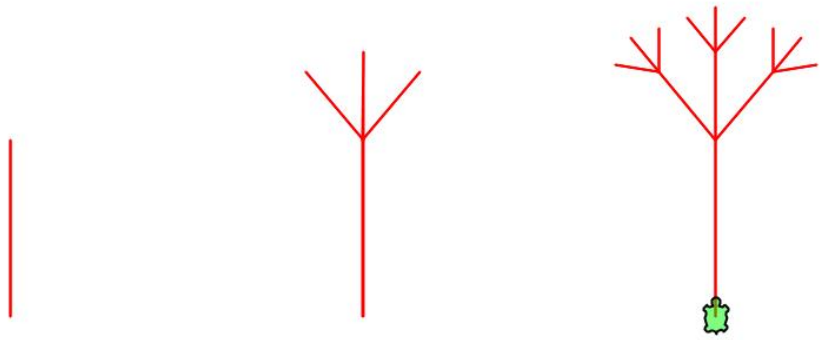

FIGURE 4 – Exemples d'arbre d'ordre 0 (à gauche), 1 (au milieu) et 2 (à droite).

Exercice 5. Approximation de pi

L'objectif de cet exercice est de tracer deux fractacles qui "convergent" vers le tracé d'un cercle (cf Figure 5) afin de pouvoir mesurer π.

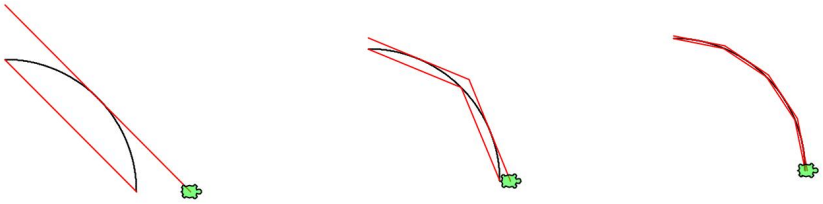

FIGURE 5 – En noir, au milieu des 2 autres tracés : le "vrai" morceau de cercle de rayon r. Les tracés à "l'intérieur" du cercle sont obtenus en appelant morceauCercleInf(r,0,90,0) (à gauche), morceauCercleInf(r,0,90,1) (au milieu) et morceauCercleInf(r,0,90,2) (à droite). Les tracés à "l'exterieur" du cercle sont obtenus en appelant morceauCercleSup(r,0,90,0) (à gauche), morceauCercleSup(r,0,90,1) (au milieu) et morceauCercleSup(r,0,90,2) (à droite).

Question 5.1.

Ecrire une méthode `public void morceauCercleInf(double r, double a1, double a2, int n)` permettant de tracer des morceaux de cercles semblables à ceux dessinés Figure 5. On pourra faire appel à la méthode `bob.setPosition(double x, double y)` qui déplace en ligne droite Bob depuis sa position actuelle vers la position (x, y), sans changer sa direction ni la position du crayon (en laissant un trait si le crayon est au contact du sol)

Question 5.2.

Ecrire une méthode `public void morceauCercleSup(double r, double a1, double a2, int n)` permettant de tracer des morceaux de cercles semblables à ceux dessinés Figure 5.

Question 5.3.
En utilisant la méthode `double d = bob.Distance(double x, double y)` qui calcule la distance entre la position actuelle de Bob et (x, y), modifiez les deux méthodes précédentes pour qu'elles retournent la longueur totale du tracé effectué.

Question 5.4.
Ecrire grâce aux méthodes précédentes une méthode `double approxPi(double e)` qui calcule une approximation de π à e près, c'est à dire qui retourne une valeur x telle que $|x - \pi| \leq e$.

Exercice 6. Dragon

Question 6.1.
Ecrire une méthode pour tracer des dragons à ceux dessinés Figure (la taille des tracés n'est pas importante ici). Vous pouvez utiliser des méthodes auxiliaires, ou ayant des paramètres supplémentaires.

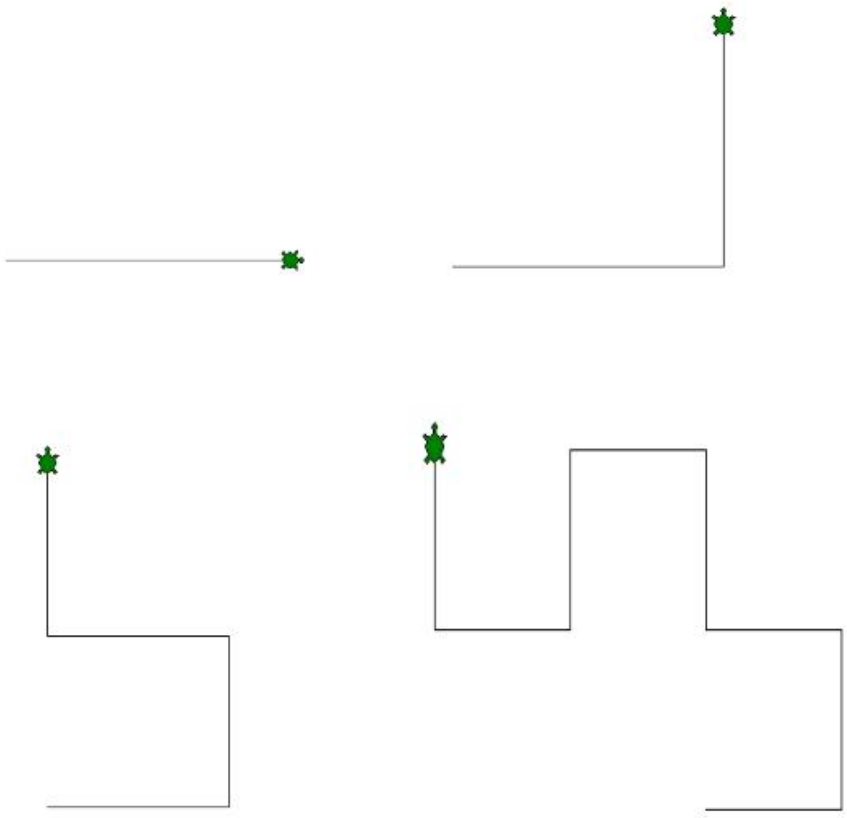

FIGURE 6 – Exemples de dragons d'ordre 0, 1, 2 et 3.

Exercice 7. Triforce (ou presque) (Sierpinski)

Question 7.1.
Ecrire une méthode pour tracer des figures semblables à celles dessinées Figure (la taille totale du triangle englobant ne varie pas).

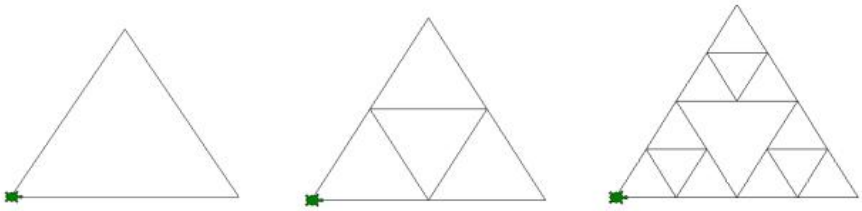

FIGURE 7 – Exemples de Sierpinski à l'ordre 0, 1 et 2.