Exercice 1 – Visualiser l'effet d'un pool de threads : la courbe de Von Koch

La courbe de Von Koch est une fractale dont la construction repose sur le principe récursif suivant : partant d'un segment limité par deux points M et N, on divise ce segment en trois parties égales. Soient A et C les points créés par cette division, on construit entre ces deux points un point B tel que A, B, C est un triangle équilatéral. Puis on répète l'opération sur chacun des segments ainsi créés. Le principe est illustré par la Figure 1.

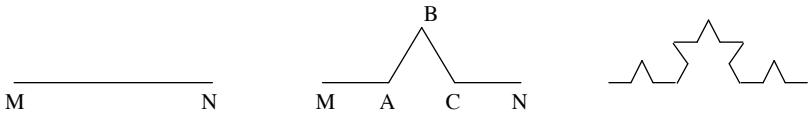

FIGURE 1 – Découpage d'un segment

La condition d'arrêt de la récursivité est liée à la taille des segments créés : lorsque la longueur de ceux-ci devient inférieure à une taille déterminée, on arrête le découpage. L'ensemble de classes ci-dessous propose une implémentation de ce tracé. La figure initiale est un triangle, et chaque côté du triangle subit une transformation.

```
import graphic.Window;
import java.awt.Point;

public class Segment {
   private Window f;
   private final Point m, n;

   public Segment (Window f, Point m, Point n) {
      this.f = f;
      this.m = m;
      this.n = n;
   }

   public static double longueur(Point m, Point n) {
      return Math.sqrt( Math.pow (m.y - n.y, 2.0) + Math.pow (m.x - n.x, 2.0) );
   }

   public static void tracer (Window f, Point m, Point n) {
      f.plotLine(m, n);
   }
}

import graphic.Window;
import java.awt.Point;

public class Cote {
   private final static double LG_MIN = 8.0;
   private final Window f;
   private final Point m, n;

   public Cote (Window f, Point m, Point n) {
      this.f = f;
      this.m = m;
      this.n = n;
   }

   public void tracer () {
      final double xa, ya, xb, yb, xc, yc;

      if ( Segment.longueur(m, n) > LG_MIN ) {
         xa = (2 * m.x + n.x) / 3.0;
         xc = (m.x + 2 * n.x) / 3.0;
         ya = (2 * m.y + n.y) / 3.0;
         yc = (m.y + 2 * n.y) / 3.0;
         xb = xa + ( xc - xa - (Math.sqrt(3.0) * (yc - ya)) ) / 2.0;
         yb = ya + ( yc - ya + (Math.sqrt(3.0) * (xc - xa)) ) / 2.0;

         Point a = new Point();
         a.setLocation(xa, ya);
         Point b = new Point();
```

```
         b.setLocation(xb, yb);
         Point c = new Point();
         c.setLocation(xc, yc);

         new Cote(f, m, a).tracer();
         new Cote(f, a, b).tracer();
         new Cote(f, b, c).tracer();
         new Cote(f, c, n).tracer();
      }
      else {
         Segment.tracer(f, m, n);
      }
   }
}
```

```
import graphic.Window;
import java.awt.Point;

public class VonKochMono {
   private final static double LG_MIN = 8.0;
   Window f;

   public VonKochMono (Window f, Point a, Point b, Point c) {
      this.f = f;
      new Cote(f, b, a).tracer();
      new Cote(f, a, c).tracer();
      new Cote(f, c, b).tracer();
   }
}
```

```
import graphic.Window;
import java.awt.Point;

public class TestVonKoch {

   public static void main (String[] args) {
      final int XMAX = 400;
      final int YMAX = 400;

      final Window f = new Window(XMAX, YMAX, "von Koch");

      Point v = new Point();
      v.setLocation(XMAX/2, YMAX/8);

      Point u = new Point();
      u.setLocation(v.x - (3.0 * XMAX/10.0), v.y + (0.5196 * XMAX));

      Point w = new Point();
      w.setLocation(v.x + (3.0 * XMAX/10.0), u.y);

      VonKochMono vk = new VonKochMono(f, u, v, w);

   }
}
```

Question 1

Compilez le programme précédent et exécutez le.

Question 2

Proposez une version multi-threads de ce programme dans laquelle chaque ligne affichée dans la fenêtre graphique est tracée par un thread différent.

Dans cette version, le nombre de threads créés par le programme est lié au nombre de lignes tracées. Nous voulons modifier le programme pour maîtriser le nombre de threads utilisés. On ne s'intéresse dans un premier temps qu'aux threads créés par le traitement d'un côté du triangle initial (les trois threads qui lancent chacun le traitement d'un côté continuent à exécuter exactement une tâche).

Question 3

Proposez une nouvelle implémentation multi-threads de la classe `Cote`, dans laquelle la création des threads se fait par un appel à une méthode de la classe `Executors`. Regardez ce qu'il se passe à l'exécution lorsqu'on limite la taille du pool à un unique thread, puis avec des pools de threads de tailles différentes.

Question 4

Modifiez l'implémentation de manière à ce que *toutes* les intstances de `Cote` soient exécutées dans un unique pool de threads.