Exercice A.1 Observation d'un gain Le répertoire `TP/Monte-Carlo` disponible dans l'archive `TP_A.tgz` fournit le code de l'exercice I.3 (page 2) et celui d'une solution parallèle *avec 4 threads*.

Question 1. Trouvez, par tâtonnement, un nombre de tirages qui nécessite, pour le programme séquentiel, environ 30 secondes d'exécution sur votre machine.

Question 2. Quel est le temps de calcul du programme parallèle proposé *pour le même nombre de tirages* ?

Question 3. Le gain observé (c'est-à-dire le rapport entre le temps de calcul parallèle sur le temps de calcul séquentiel) vous paraît-il cohérent avec le nombre de processeurs disponibles ?

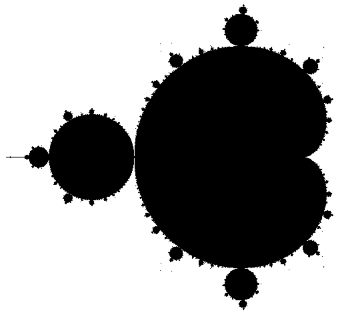
Exercice A.2 Partage statique et dynamique d'un dessin L'ensemble de Mandelbrot est une fractale définie comme l'ensemble des points c du plan complexe pour lesquels la suite définie par : $z_{n+1} = z_n^2 + c$ avec $z_0 = 0$ ne tend pas, en module, vers l'infini. Si nous reformulons cela sans utiliser les nombres complexes, en remplaçant z_n par le couple de réels (x_n, y_n) et c par le couple (a, b), alors nous obtenons :

$$x_{n+1} = x_n^2 - y_n^2 + a \text{ et } y_{n+1} = 2.x_n.y_n + b \text{ avec } x_0 = y_0 = 0$$

la condition devenant que ni x_n, ni y_n ne tendent vers l'infini (en valeur absolue).

Dans la pratique, nous allons calculer les premiers termes de cette suite pour un point (a, b) fixé. Si la valeur de $x_n^2 + y_n^2$ dépasse 4, nous considérerons (de manière arbitraire) que la suite diverge, et donc que le point (a, b) n'est pas dans l'ensemble. En revanche si la valeur $x_n^2 + y_n^2$ ne dépasse 4 pour aucun des termes calculés, nous considérerons que la suite ne diverge pas, et donc que le point (a, b) est dans l'ensemble. Ceci nous conduit à utiliser la méthode suivante, qui détermine si oui ou non le point (a, b) est dans l'ensemble de Mandelbrot lorsque l'on examine les `max` premiers éléments de la suite (x_n, y_n).

```java
static boolean mandelbrot(double a, double b, int max) {
  double x = 0;
  double y = 0;
  for (int t = 0; t < max; t++) {
    if (x*x + y*y > 4.0) return false; // Le point est blanc
    double nx = x*x - y*y + a;
    double ny = 2*x*y + b;
    x = nx;
    y = ny;
  }
  return true; // Le point est noir
}
```



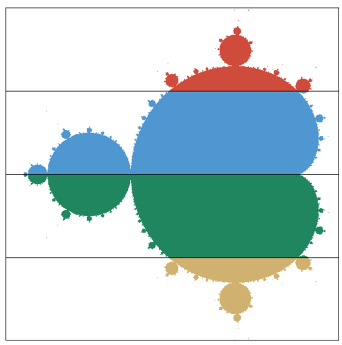
Le programme `Mandelbrot.java` disponible dans l'archive `TP_A.tgz` est donné sur la figure 3. Il calcule et affiche l'ensemble de Mandelbrot situé dans la région carrée comprise entre $(-1.5, -1)$ en bas à gauche et $(0.5, 1)$ en haut à droite. Le nombre de pixels par ligne (et par colonne) est fixé à `taille=500` alors que le nombre maximal d'itérations vaut `max=20_000`. Le programme indique également le temps de calcul de l'image. Vous pouvez annuler l'affichage de l'image en supprimant l'instruction `image.show()`.

Question 1. Modifiez la valeur de `max` pour que le calcul de l'image avec ce programme dure environ une trentaine de secondes.

Question 2. Partagez ensuite le calcul de l'image sur 4 threads, chacun calculant un quart de l'image, *sans modifier les fonctions* ***`colorierPixel()`*** *et* ***`mandelbrot()`***.

Question 3. Quel est le gain, en terme de temps de calcul de l'image complète, par rapport à la version séquentielle ? Quel est le temps de calcul de chacun des quatre threads ?

Question 4. Attribuez à présent à la volée les lignes de l'image aux 4 threads au fur et à mesure qu'ils sont disponibles. Quel est le nouveau gain, en terme de temps de calcul, par rapport à la version séquentielle ?



Exercice A.3 L'état du cobaye (suite) Nous poursuivons l'exercice I.5. Pour chacun des programmes du répertoire `TP/Cobaye`, devinez l'affichage produit par chaque appel à la méthode **`Affiche()`** puis vérifiez votre hypothèse en lançant une exécution.

```java
import java.awt.Color;

public class Mandelbrot {
    final static Color noir = new Color(0, 0, 0);
    final static Color blanc = new Color(255, 255, 255);
    final static int taille = 500;   // nombre de pixels par ligne (et par colonne)
    // Il y a donc taille*taille pixels blancs ou noirs à déterminer
    final static Picture image = new Picture(taille, taille);

    final static double xc = -.5 ;
    final static double yc = 0 ; // Le point (xc,yc) est le centre de l'image
    final static double region = 2;
    // La région du plan considérée est un carré de côté égal à 2.
    // Elle s'étend donc du point (xc - 1, yc - 1) au point (xc + 1, yc + 1)
    // c'est-à-dire du point (-1.5, -1) en bas à gauche au point (0.5, 1) en haut
    // à droite

    final static int max = 40_000;
    // C'est le nombre maximum d'itérations pour déterminer la couleur d'un pixel

    public static void main(String[] args)  {
        final long startTime = System.nanoTime();
        final long endTime;
        for (int i = 0; i < taille; i++) {
            for (int j = 0; j < taille; j++) {
                colorierPixel(i,j);
            }
        }
        endTime = System.nanoTime();
        final long duree = (endTime - startTime) / 1_000_000 ;
        System.out.println("Durée = " + (long) duree + " ms.");
        image.show();
    }

    // La fonction colorierPixel(i,j) colorie le pixel (i,j) de l'image
    public static void colorierPixel(int i, int j) {
        double a = xc - region/2 + region*i/taille;
        double b = yc - region/2 + region*j/taille;
        // Le pixel (i,j) de l'image correspond au point (a,b) de la région
        if (mandelbrot(a, b, max)) image.set(i, j, noir);
        else image.set(i, j, blanc);
    }

    // La fonction mandelbrot(a, b, max) détermine si le point (a,b) est noir
    public static boolean mandelbrot(double a, double b, int max) {
        double x = 0;
        double y = 0;
        for (int t = 0; t < max; t++) {
            if (x*x + y*y > 4.0) return false; // Le point (a,b) est blanc
            double nx = x*x - y*y + a;
            double ny = 2*x*y + b;
            x = nx;
            y = ny;
        }
        return true; // Le point (a,b) est noir
    }
}


/*
  $ javac Mandelbrot.java
  $ java Mandelbrot
  Durée = 17831 ms.
  $ make
  $ java -jar Mandelbrot.jar
  Durée = 17869 ms.
*/
```

FIGURE 3 – Fractale de Mandelbrot