Outline

- OOP analysis examples
  - Random objects vs Math.random
  - DrawingPanel objects vs StdDraw

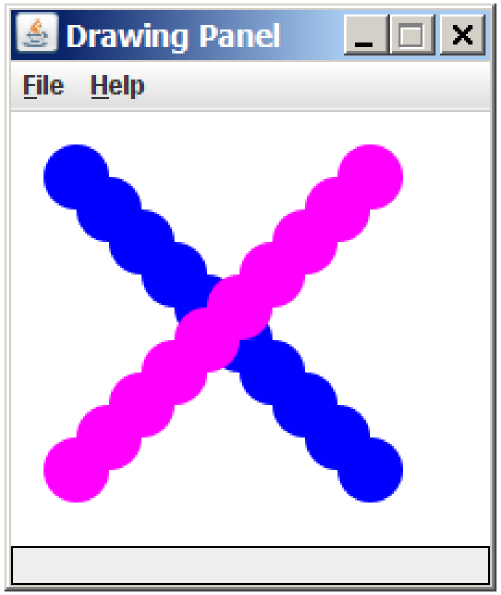
StdDraw

- Java graphics by nature is OOP
- StdDraw is a library to hide OOP programming complexity (No objects)
  - Just as
    Math.random is a delegation to a single Random object
    StdDraw is a delegation to a graphics window

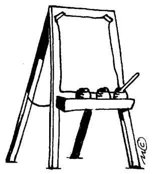
`DrawingPanel` Design

Developed for the Back to Basics Textbook, using an OOP approach. Two key classes (types of objects):

- `DrawingPanel`: A window on the screen.
  - Not part of Java; provided by the textbook.
- `Graphics`: A "pen" to draw shapes and lines on a window.
  - from Java standard class library



DrawingPanel



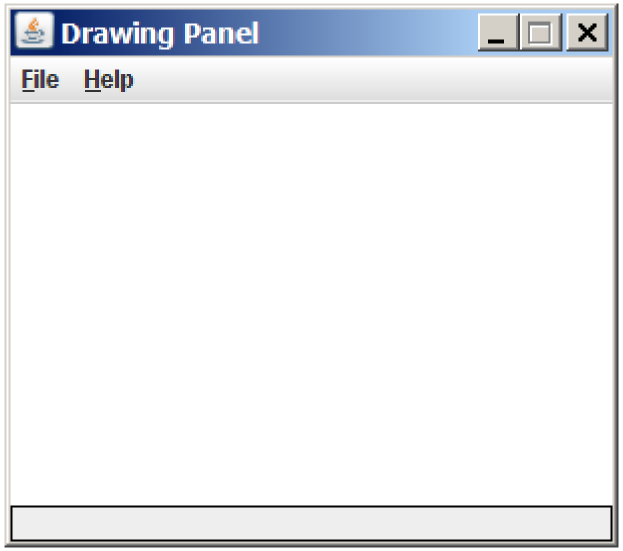
A "Canvas" object that represents windows/drawing surfaces

- To create a window:

  `DrawingPanel` **name** `= new DrawingPanel(`**width, height**`);`

  Example:

```
DrawingPanel panel = new DrawingPanel(300, 200);
```



See SimplePanels.java

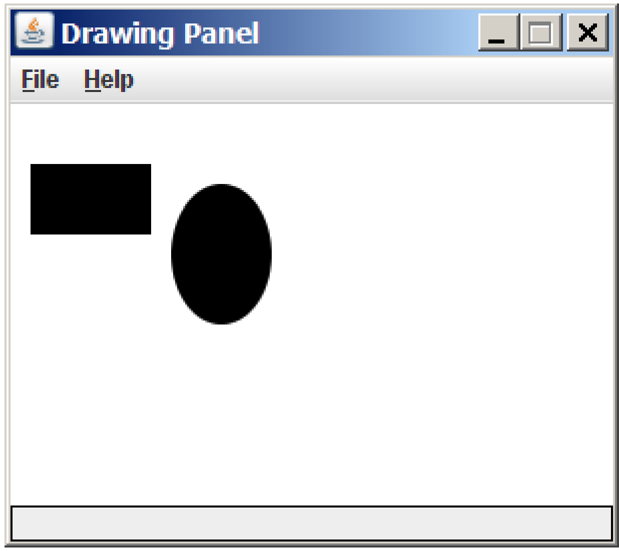
Graphics

- DrawingPanel draws shapes / lines /characters /imagines using another object of type `Graphics`.
  - *Graphics: Your painting toolset: "Pen" or "paint brush" objects to* ***remember state*** *and draw lines and shapes; fonts for character, …*
    - Access it by calling `getGraphics` on a `DrawingPanel` object.

```
Graphics g = panel.getGraphics()
```

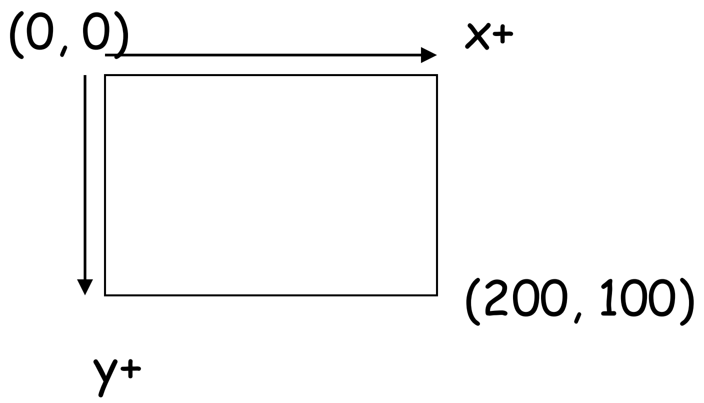
  - Draw shapes by calling methods on the `Graphics` object.

```
g.setColor(Color.BLACK);
g.fillRect(10, 30, 60, 35);
g.fillOval(80, 40, 50, 70);
```



Graphics Coordinate System

- Each (x, y) position is a *pixel* ("picture element").
- Position (0, 0) is at the window's top-left corner.
  - x increases rightward and the y increases downward.
- A rectangle from (0, 0) to (200, 100) looks like this:



Some `Graphics` Methods

Method name	Description
`g.setColor(`**Color**`);`	set `Graphics` to paint any following shapes in the given color
`g.drawLine(`**x1**, **y1**, **x2**, **y2**`);`	line between points (*x1*, *y1*), (*x2*, *y2*)
`g.drawOval(`**x**, **y**, **width**, **height**`);`	outline largest oval that fits in a box of size *width* * *height* with top-left at (*x*, *y*)
`g.drawRect(`**x**, **y**, **width**, **height**`);`	outline of rectangle of size *width* * *height* with top-left at (*x*, *y*)
`g.drawString(`**text**, **x**, **y**`);`	text with bottom-left at *(x, y)*
`g.fillOval(`**x**, **y**, **width**, **height**`);`	fill largest oval that fits in a box of size *width* * *height* with top-left at (*x*, *y*)
`g.fillRect(`**x**, **y**, **width**, **height**`);`	fill rectangle of size *width* * *height* with top-left at (*x*, *y*)

`See SimplePanels.java`

http://download.oracle.com/javase/6/docs/api/java/awt/Graphics.html

Outline

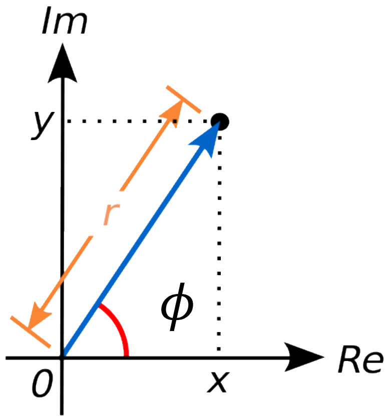
- OOP analysis examples
  - Random objects vs Math.random
  - Complex numbers and fractal graphics

Complex Numbers

- A complex number (a + b*i*) is a quintessential mathematical abstraction
  - (a + bi) + (c + di) = a + c + (b + d) i
  - (a + bi) x (c + di) = ac - bd + (ad + bc)i

$$a = 3 + 4i,\ \ b = -2 + 3i$$
$$a + b = 1 + 7i$$
$$a * b = -18 + i$$
$$|\,a\,| = 5$$

- A main power of complex numbers comes from Euler's formula

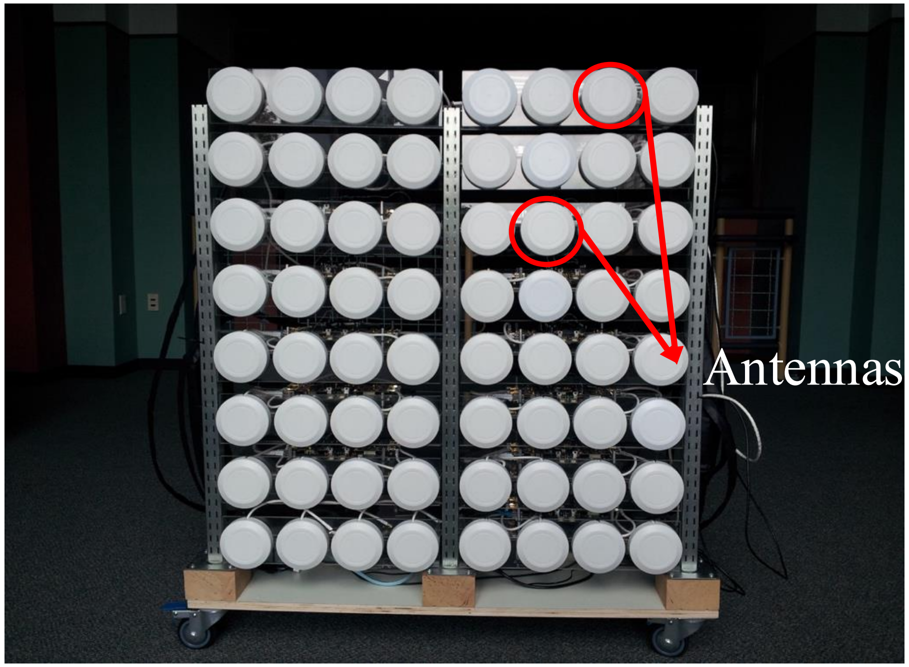


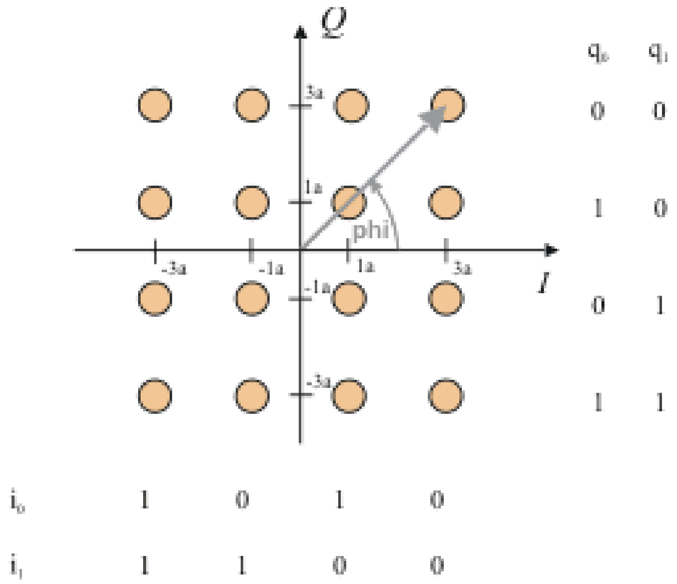
$$\mathrm{r}\,[\cos(\phi) + \sin(\phi)i] \quad = \mathrm{r}\ \mathrm{e}^{i\phi}$$

$$\mathrm{r}_1\, e^{\varphi_1} * \mathrm{r}_2\, e^{\varphi_2} \ = \mathrm{r}_{12} e^{\varphi_1 + \varphi_2}$$

Complex Numbers are Widely Used

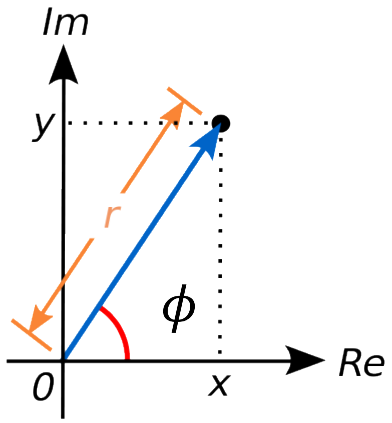
- Control theory and Laplace transforms
- Quantum mechanics and Hilbert spaces
- Fractals
- Signal processing and Fourier analysis





A `Complex` Class

- Design questions:
  - State: what field(s) do we need to represent the state of a complex number?
    - Design 1
      - `re`, a number representing the real part
      - `im`, a number representing the imaginary part
    - Design 2
      - `r`, a number representing the distance to origin
      - `theta`, a number representing the angle



A `Complex` Class

- Design questions:
  - Behaviors: what are some common behaviors of a complex number?
    - a `Complex` constructor, to set up the object
    - A `abs` method, to return the distance (magnitude)
    - a `toString` method: Return a string description of a complex number
    - Mathematical operations such as +, -, *
      - a `plus` method: Add current complex number with another complex number
      - a `times` method: Multiply current complex number with another complex number
      - …

The `Complex` Class: Design Question

```
public class Complex {

   private double re;
   private double im;

   public Complex(double real, double imag) {
      re = real;
      im = imag;
   }

   public ??  plus(Complex b) {
      …
   }
…
```

- What is the return type of plus?
- Should plus change the state of the number, e.g.,
  Complex c1 = new Complex(1, 1);
  Complex c2 = new Complex(2, 1);
  c1.plus(c2); // c1 changes to (3, 2)?

The Consistency (Familiarity) Design Principle

- Basic idea when Defining the behaviors of a type A:
  - Think if there is a well-known type B. If so, make A's behaviors consistent w/ B
- Suppose a, b, and c are standard numbers (Complex numbers are numbers after all)
  - Does a + b (think a.+(b)) change a?
    - no
  - What is the return of a + b (think a.+(b))?
    - The value of a + b so that we can write a + b + c
- Complex.plus behavior design:

```
public Complex plus(Complex b) {
      double real = re + b.re;
      double imag = im + b.im;
      return new Complex(real, imag);
}
```

Complex.java

```
public class Complex {

   private double re;
   private double im;

   public Complex(double real, double imag) {
      re = real;
      im = imag;
   }

   public String toString() { return re + " + " + im + "i"; }

   public double abs() { return Math.sqrt(re*re + im*im); }

   public Complex plus(Complex b) {
      double real = re + b.re;
      double imag = im + b.im;
      return new Complex(real, imag);
   }

   public Complex times(Complex b) {
      double real = re * b.re – im * b.im;
      double imag = re * b.im + im * b.re;
      return new Complex(real, imag);
   }
}
```

instance variables

constructor

refers to b's instance variable

creates a Complex object, and returns a reference to it

methods

Immutability: Advantages and Disadvantages

- Consistency w/ primitive types leads to **immutable** data types: object's state does not change once constructed.
  - Example: Complex object, String.
- Advantages.
  - Easier for debugging.
    - Avoid aliasing bugs.
  - Safety:
    - Limits scope of code that can change values.
    - Pass objects around without worrying about modification.
- Disadvantage.
  - New object must be created for every value.

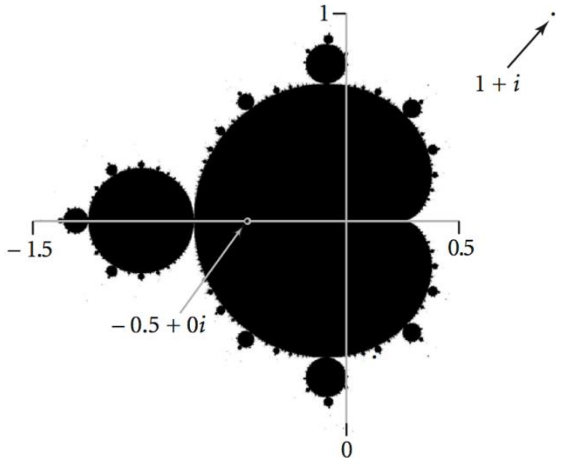
A Simple Client

```java
public static void main(String[] args) {
   Complex a = new Complex( 3.0, 4.0);
   Complex b = new Complex(-2.0, 3.0);
   Complex c = a.times(b);
   System.out.println("a = " + a.toString() );
   System.out.println("b = " + b.toString() );
   System.out.println("c = " + c.toString() );
}
```

```
% java TestClient
a = 3.0 + 4.0i
b = -2.0 + 3.0i
c = -18.0 + 1.0i
```

A Complex Client: Mandelbrot Set

- Mandelbrot set. A set of complex numbers.
- Plot.
  - Plot (x, y) black if $z = x + y\,i$ is in the set, and white otherwise.



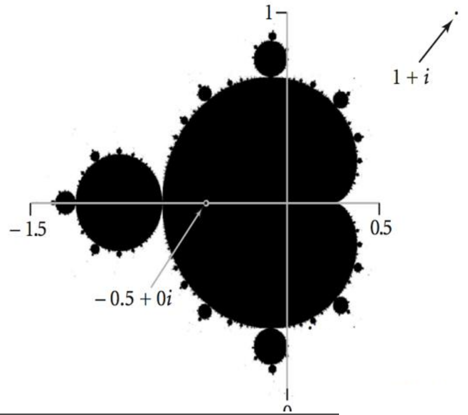
- Can be used to model complex rugged shapes such as uneven clouds, contours of mountains, winding riverbeds, arts, …

http://users.math.yale.edu/mandelbrot/

A Complex Client: Mandelbrot Set

Mandelbrot set. Is complex number z_0 in the set?

- Iterate $z_{t+1} = (z_t)^2 + z_0$.
- If $|z_t|$ diverges to infinity, then z_0 is not in set; otherwise z_0 is in set.



t	z_t
0	-1/2 + 0i
1	-1/4 + 0i
2	-7/16 + 0i
3	-79/256 + 0i
4	-26527/65536 + 0i
5	-1443801919/4294967296 + 0i

z = -1/2 is in Mandelbrot set

t	z_t
0	1 + i
1	1 + 3i
2	-7 + 7i
3	1 - 97i
4	-9407 - 193i
5	88454401 + 3631103i

z = 1 + i not in Mandelbrot set

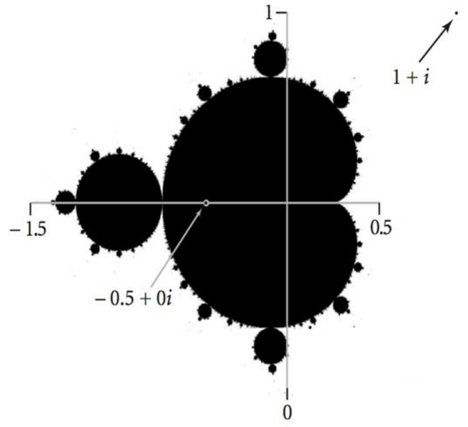
Testing Point in Mandelbrot Set

Practical issues.

- Cannot iterate infinitely many times.

Approximate solution.

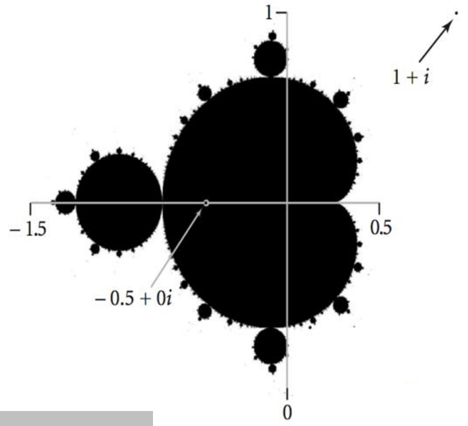
- Fact: if $|z_t| > 2$ for any t, then z not in Mandelbrot set.
- Pseudo-fact: if $|z_{255}| < 2$ then z "likely" in Mandelbrot set.



Testing Point in Mandelbrot Set

Our Mandelbrot test:

Returns the number of iterations to check if z0 is in Mandelbrot



```
public static int mand(Complex z0) {
   final int max = 255;
   Complex z = z0;
   for (int t = 0; t < max; t++) {
      if (z.abs() > 2.0) return t;
      z = z.times(z).plus(z0);

   }
   return max;
}
```

$z = z^2 + z_0$

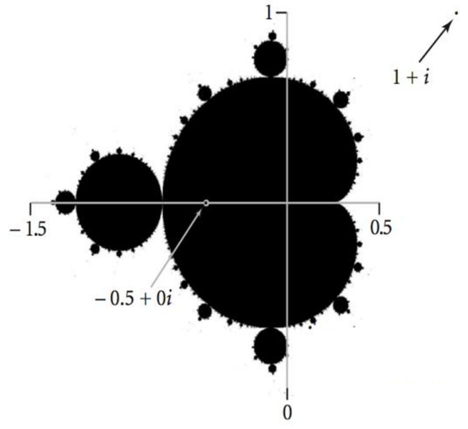
Plotting Mandelbrot Set

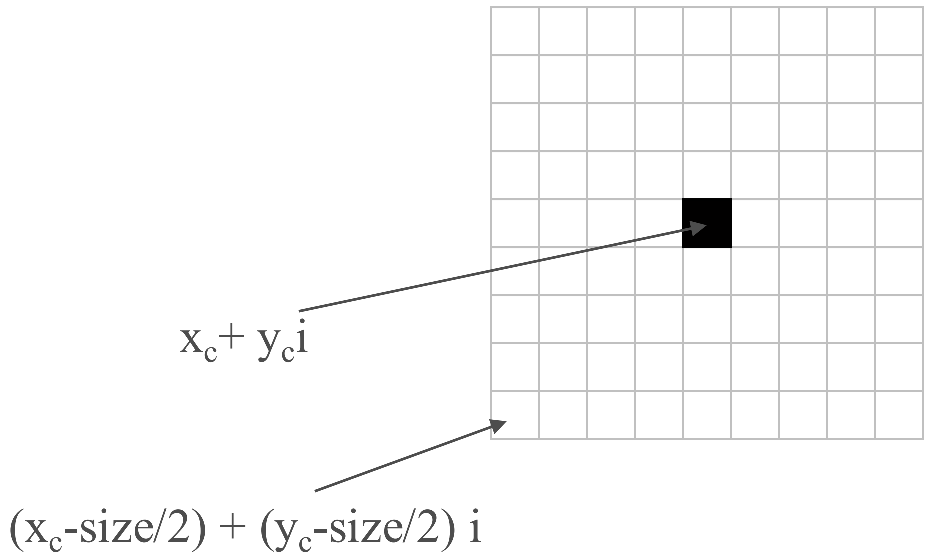
Practical issues.

- Cannot plot infinitely many points.

Display technique.

- User specifies center, size
- Program maps the points on the N-by-N drawing panel to center, size





Each grid has length size/N

Plotting Mandelbrot Set (DrawingPanel)

Plot the Mandelbrot set in gray scale.

MandelbrotDrawingPanel.java

```
public static void main(String[] args) {
   double xc   = Double.parseDouble(args[0]);
   double yc   = Double.parseDouble(args[1]);
   double size = Double.parseDouble(args[2]);

   DrawingPanel panel = new DrawingPanel(N, N);
   Graphics g = panel.getGrahics();
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         double x0 = xc - size/2 + size*i/N;
         double y0 = yc - size/2 + size*j/N;
         Complex z0 = new Complex(x0, y0);
         int gray = mand(z0);
         Color color = new Color(gray, gray, gray);
         g.setColor( color );
         g.drawLine(i, N-1-j, i, N - 1 - j);

      } // end of for
   } // end of for
}
```

(0, 0) is upper left

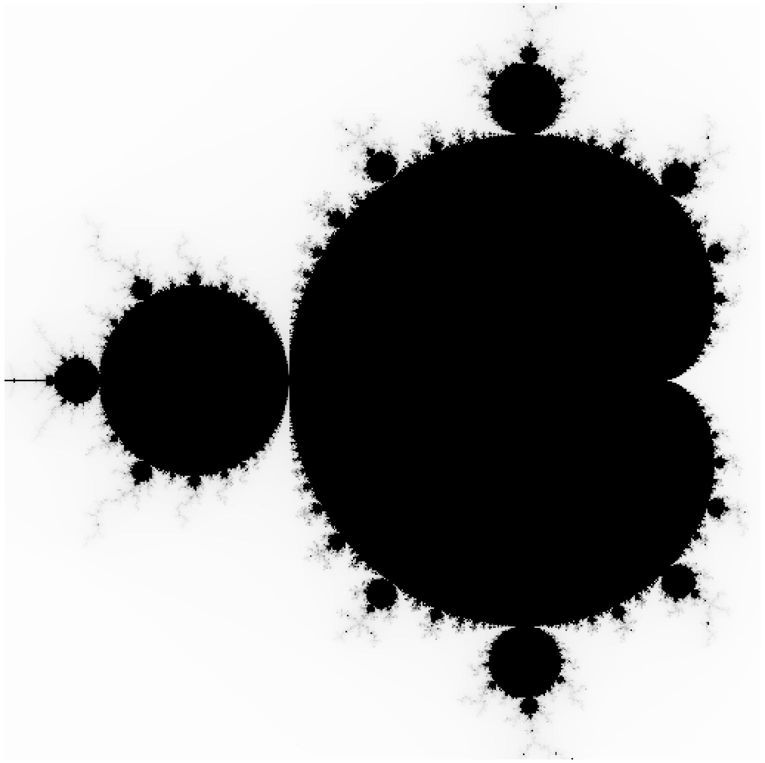
Mandelbrot Set

```
% java MandelbrotDrawingPanel -.5 0 2
```

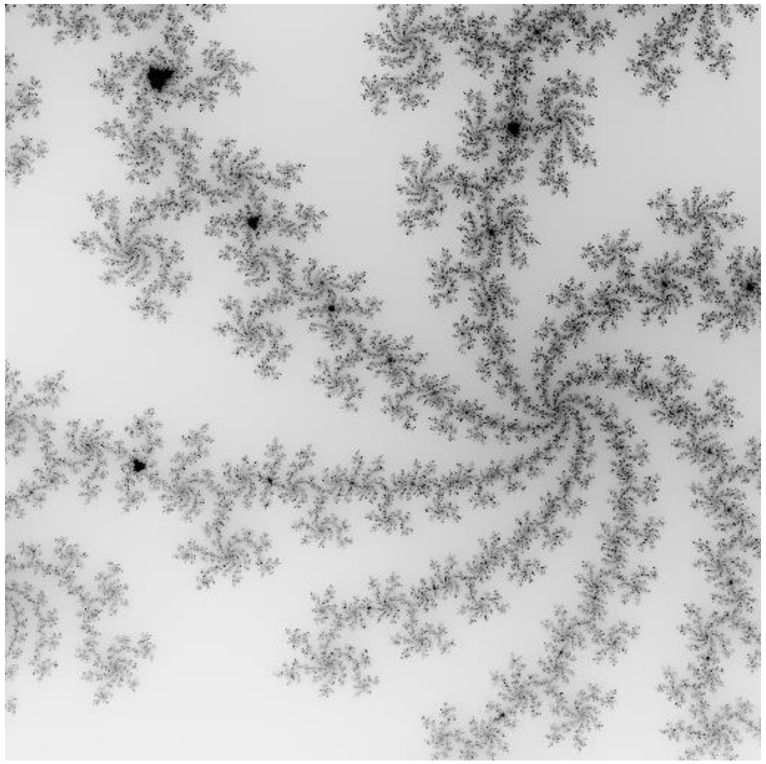
```
% java MandelbrotDrawingPanel .1045 -.637 .01
```

Mandelbrot Set

```
% java MandelbrotDrawingPanel -.5 0 2
```



```
% java MandelbrotDrawingPanel .1045 -.637 .01
```



Mandelbrot Set

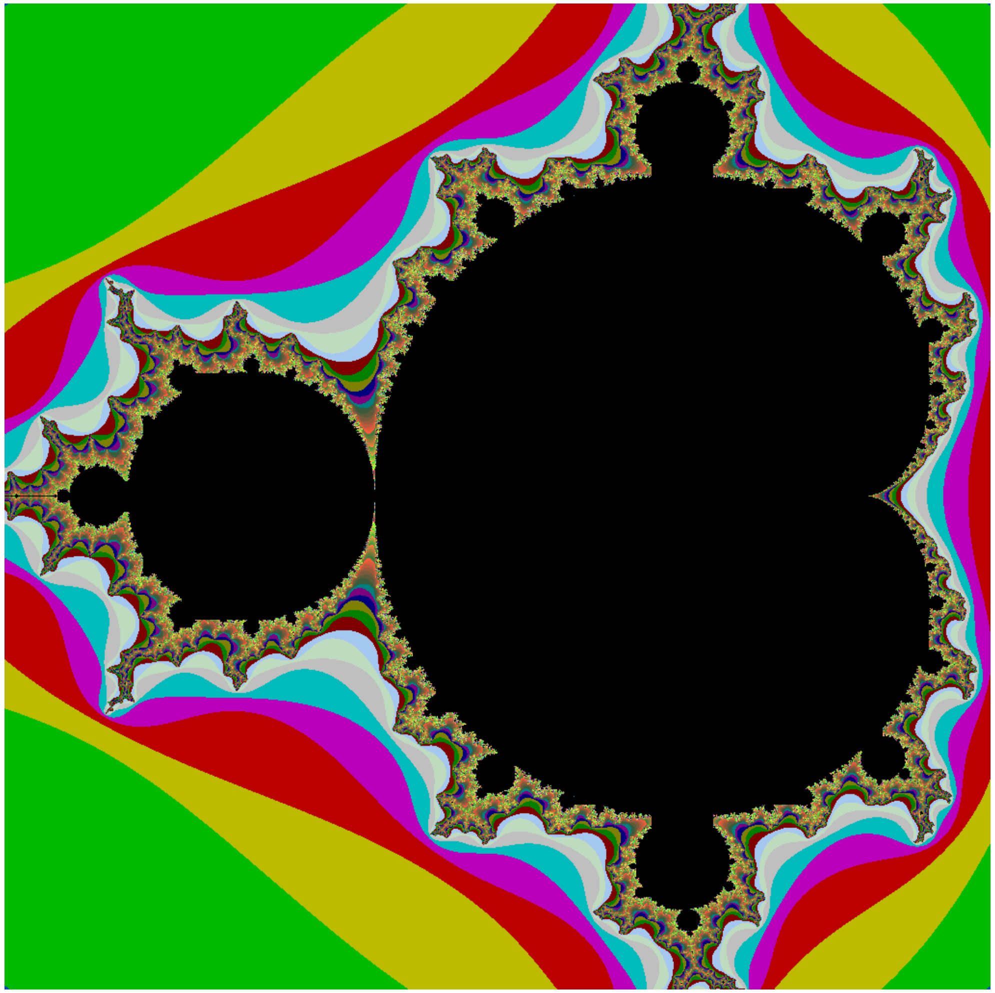
Plot the Mandelbrot set in color using a color mapping table.

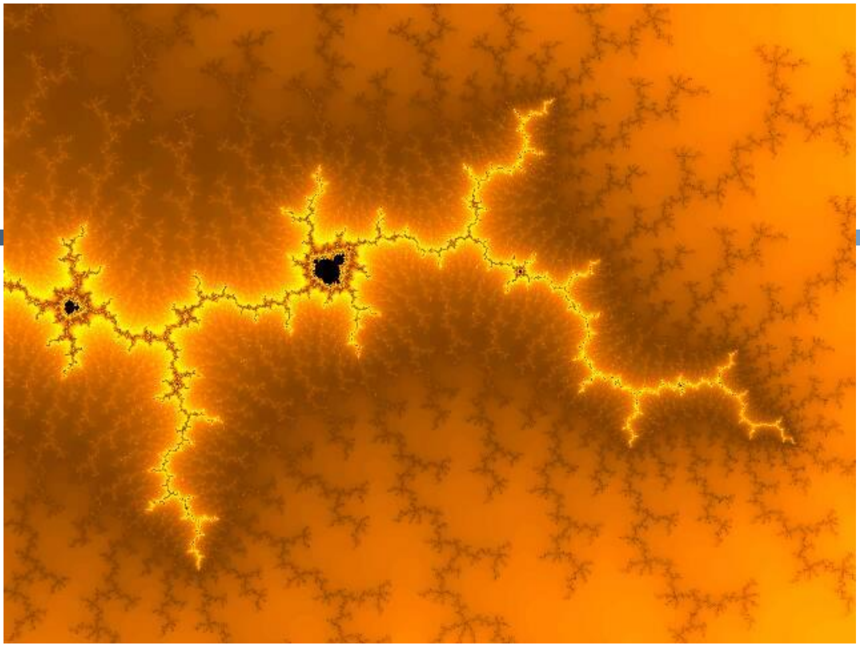
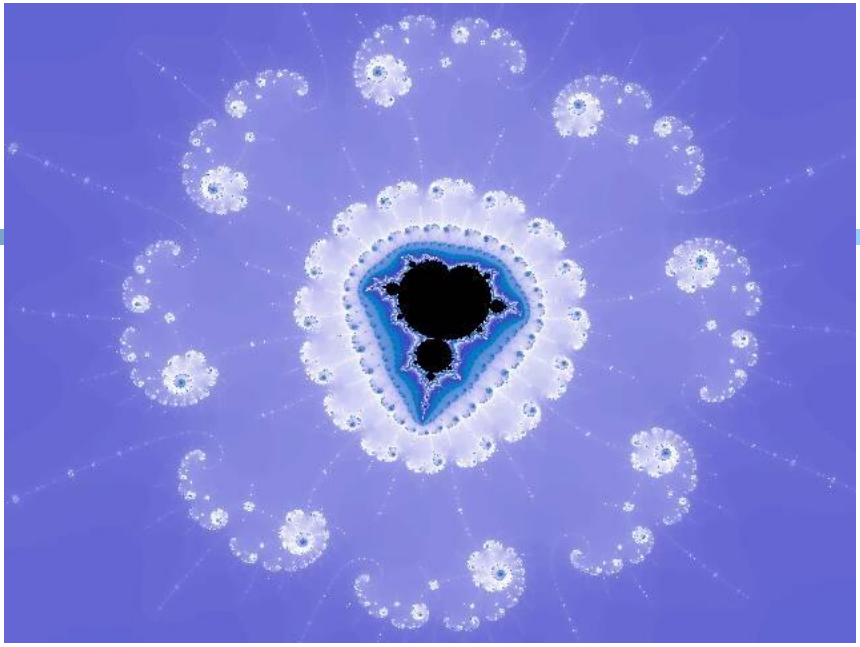
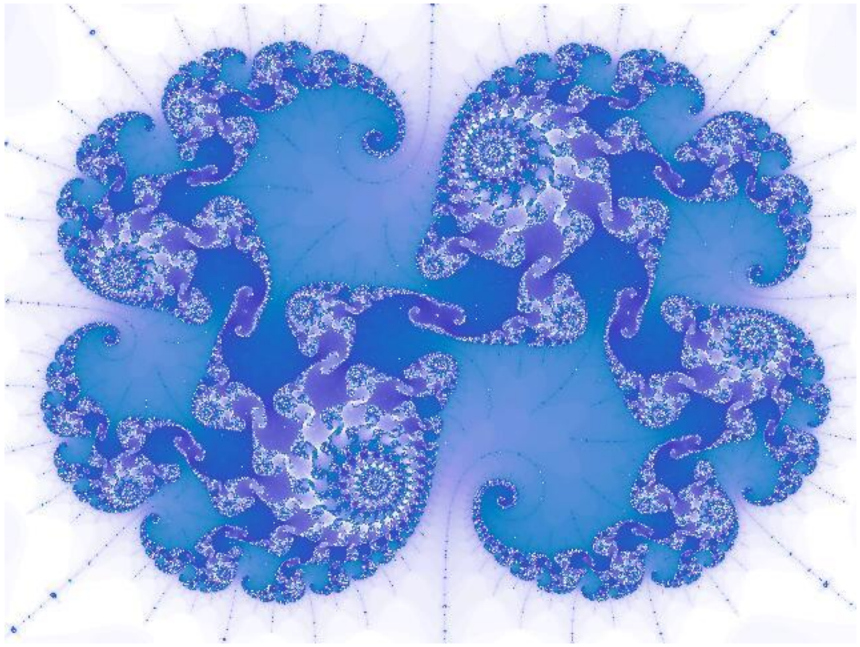
```
% java MandelbrotDrawingPanelColor -.5 0 2
```

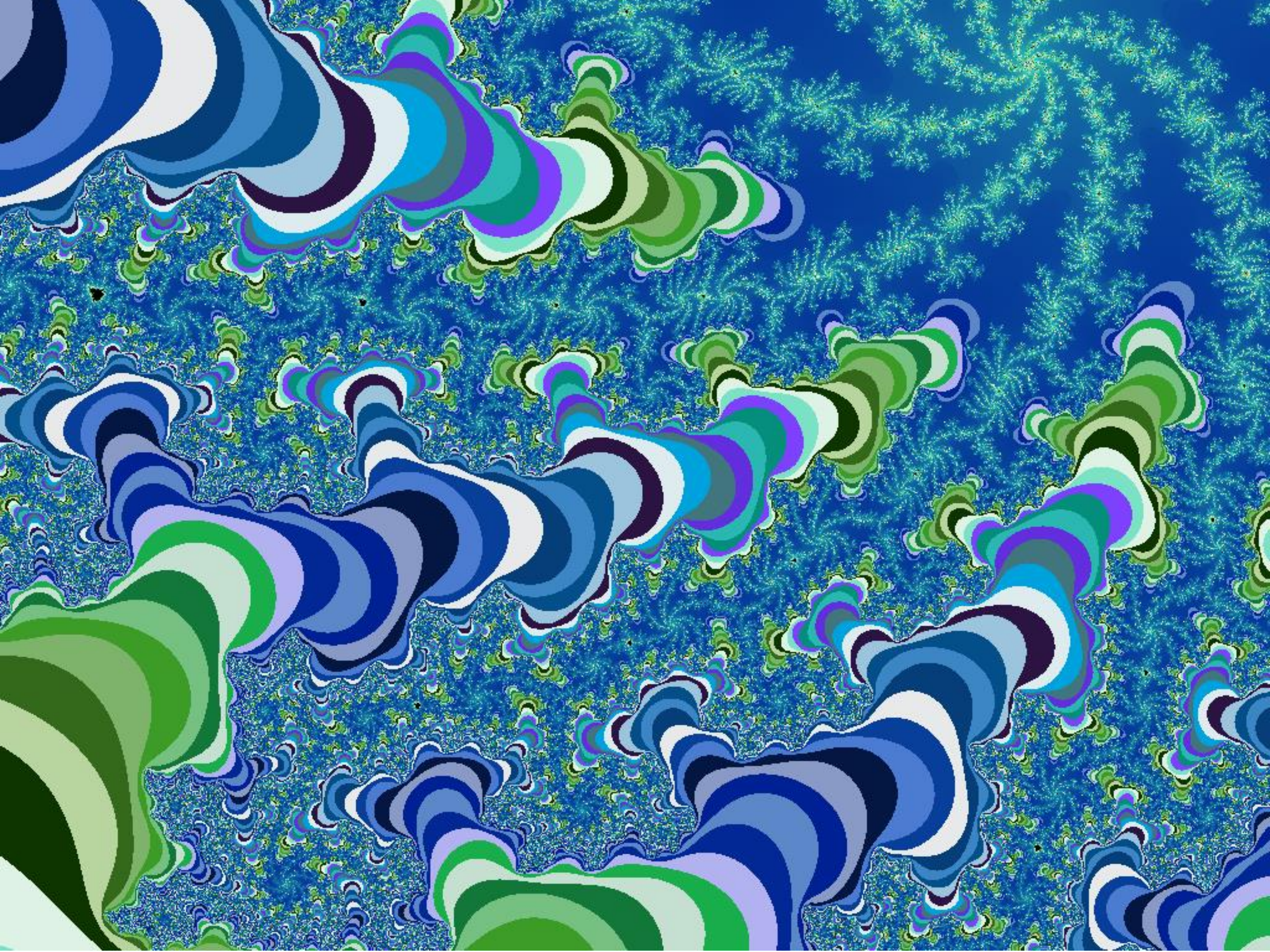
Mandelbrot Set

Plot the Mandelbrot set in color using a color mapping table.

```
% java MandelbrotDrawingPanelColor -.5 0 2
```



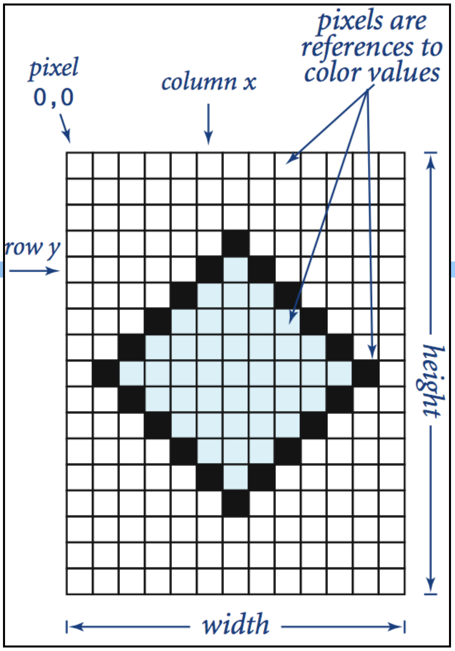






Mandelbrot Set

- See video
  https://www.bilibili.com/video/BV1ci4y1G7D2/?vd_source=4a5effc51fa44df6d45b104296de32eb

Picture Data Type



- Raster graphics. Basis for image processing.

- Set of values. 2D array of `Color` objects

`public class Picture`

	`Picture(String filename)`	*create a picture from a file*
	`Picture(int w, int h)`	*create a blank* `w`*-by-*`h` *picture*
`int`	`width()`	*return the width of the picture*
`int`	`height()`	*return the height of the picture*
`Color`	`get(int x, int y)`	*return the color of pixel (x, y)*
`void`	`set(int x, int y, Color c)`	*set the color of pixel (x, y) to* `c`
`void`	`show()`	*display the image in a window*
`void`	`save(String filename)`	*save the image to a file*

Plotting Mandelbrot Set (Picture)

Plot the Mandelbrot set in gray scale using Picture. **`MandelbrotPicture.java`**

```
public static void main(String[] args) {
   double xc   = Double.parseDouble(args[0]);
   double yc   = Double.parseDouble(args[1]);
   double size = Double.parseDouble(args[2]);

   Picture pic = new Picture(N, N); // NxN picture
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         double x0 = xc - size/2 + size*i/N;
         double y0 = yc - size/2 + size*j/N;
         Complex z0 = new Complex(x0, y0);
         int gray = mand(z0);
         Color color = new Color(gray, gray, gray);
         pic.set(i, N-1-j, color);

      } // end of for
   } // end of for
}
```

scale to screen coordinates

(0, 0) is upper left