"What makes the situation worse is that the highest level CS course I've ever taken is cs4, and quotes from the graphics group startup readme like 'these paths are abstracted as being the result of a topological sort on the graph of ordering dependencies for the entries' make me lose consciousness in my chair and bleed from the nose."

-mgrimes, Graphics problem report 134
# Heat Map

The image displays a stock heat map showing the percent change for various stock symbols. The colors range from green (positive change) to red (negative change). Here are the percent changes for different stocks:

- **PFE**: 3.09%
- **T**: 1.92%
- **PG**: 1.7%
- **MRK**: 1.49%
- **UNH**: 1.42%
- **VZ**: 1.36%
- **JNJ**: 1.1%
- **WMT**: 1.05%
- **CAT**: 1.04%
- **CVX**: 0.88%
- **KO**: 0.86%
- **MMM**: 0.82%
- **INTC**: 0.76%
- **TRV**: 0.72%
- **JPM**: 0.58%
- **MCD**: 0.54%
- **AXP**: 0.53%
- **XOM**: 0.41%
- **MSFT**: 0.12%
- **AA**: 0.11%
- **IBM**: -0.05%
- **BAC**: -0.06%
- **GE**: -0.18%
- **DD**: -0.27%
- **UTX**: -0.4%
- **DIS**: -0.48%
- **HD**: -0.55%
- **BA**: -0.91%
- **CSCO**: -1.66%
- **HPQ**: -3.58%

The heat map uses color gradients to indicate the magnitude of the percent change, with green representing positive changes and red indicating negative changes. The update values button allows for refresh or modification of the data.
Random Art
Image Manipulation
Simulation and Visualization

WatorWorld
Fractal 3D Landscape
Objects (briefly)

- **object**: An entity that contains data and behavior.
  - *data*: variables inside the object
  - *behavior*: methods called on object
    - You interact with the methods; the data is hidden in the object.
    - A **class** is a data type.

- Constructing (creating) an object:
  ```
  Type objectName = new Type(parameters);
  ```

- Calling an object's method:
  ```
  objectName . methodName (parameters) ;
  ```
We will draw graphics in Java using 3 kinds of objects:

- **DrawingPanel**: A window on the screen.
  - Not part of standard Java; provided by the authors. See class web site.

- **Graphics**: A "pen" to draw shapes and lines on a window.

- **Color**: Colors in which to draw shapes.
"Canvas" objects that represents windows/drawing surfaces

- To create a window:

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

  Example:
  ```java
  DrawingPanel panel = new DrawingPanel(300, 200);
  ```

- The window has nothing on it.
  - We draw shapes / lines on it with another object of type Graphics.
Graphics

"Pen" or "paint brush" objects to draw lines and shapes

- Access it by calling `getGraphics` on your `DrawingPanel`.
  ```java
  Graphics g = panel.getGraphics();
  ```

- Draw shapes by calling methods on the `Graphics` object.
  ```java
  g.fillRect(10, 30, 60, 35);
g.fillOval(80, 40, 50, 70);
  ```
Java class libraries, import

- **Java class libraries**: Classes included with Java's JDK.
  - organized into groups named *packages*
  - To use a package, put an *import declaration* in your program:
    ```java
    // put this at the very top of your program
    import packageName.ClassName;
    ```
- **Graphics** belongs to a package named *java.awt*
  ```java
  import java.awt.Graphics;
  ```
- To use Graphics, you must place the above line at the very top of your program, before the *public class* header.
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*.

- The rectangle from (0, 0) to (200, 100) looks like this:
# Graphics methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g.drawLine(x1, y1, x2, y2);</code></td>
<td>line between points ((x1, y1), (x2, y2))</td>
</tr>
<tr>
<td><code>g.drawOval(x, y, width, height);</code></td>
<td>outline largest oval that fits in a box of size (width \times height) with top-left at ((x, y))</td>
</tr>
<tr>
<td><code>g.drawRect(x, y, width, height);</code></td>
<td>outline of rectangle of size (width \times height) with top-left at ((x, y))</td>
</tr>
<tr>
<td><code>g.drawString(text, x, y);</code></td>
<td>text with bottom-left at ((x, y))</td>
</tr>
<tr>
<td><code>g.fillOval(x, y, width, height);</code></td>
<td>fill largest oval that fits in a box of size (width \times height) with top-left at ((x, y))</td>
</tr>
<tr>
<td><code>g.fillRect(x, y, width, height);</code></td>
<td>fill rectangle of size (width \times height) with top-left at ((x, y))</td>
</tr>
<tr>
<td><code>g.setColor(Color);</code></td>
<td>set Graphics to paint any following shapes in the given color</td>
</tr>
</tbody>
</table>
Specified as predefined `Color` class constants:

```java
Color.CONSTANT_NAME
```

where `CONSTANT_NAME` is one of:

- BLACK, BLUE,
- CYAN, DARK_GRAY,
- GRAY,
- GREEN, LIGHT_GRAY,
- MAGENTA, ORANGE,
- PINK, RED,
- WHITE, YELLOW

Or create one using `Red-Green-Blue` (RGB) values of 0-255

```java
Color name = new Color(red, green, blue);
```

- Example:
  ```java
  Color brown = new Color(192, 128, 64);
  Color burntOrange = new Color(191, 87, 0);
  ```
How many rectangles appear on the DrawingPanel when the following code is run?

```java
DrawingPanel p1 = new DrawingPanel(200, 200);
Graphics gr = new Graphics();
for(int i = 0; i < 5; i++) {
    gr.drawRect(i * 25, i * 20, 20, 50);
}
```

A. 5   B. 6   C. 20
D. None due to syntax error
E. None due to runtime error
What named color is closest to the Color object created by this code?

```java
Color mc = new Color(255, 255, 255);
```

A. Black
B. Brown
C. Gray
D. Orange
E. White
Using colors

- **Pass a** Color to Graphics object's `setColor` method
  - Subsequent shapes will be drawn in the new color.

  ```java
  g.setColor(Color.BLACK);
  g.fillRect(10, 30, 100, 50);
  g.drawLine(20, 0, 10, 30);
  g.setColor(Color.RED);
  g.fillOval(60, 40, 40, 70);
  ```

- **Pass a color to** DrawingPanel's `setBackground` method
  - The overall window background color will change.

  ```java
  Color brown = new Color(192, 128, 64);
  panel.setBackground(brown);
  ```
To draw a colored shape with an outline, first *fill* it, then *draw* the same shape in the outline color.

```java
import java.awt.Graphics; // so I can use Graphics
import java.awt.Color;

public class OutlineExample {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(150, 70);
        Graphics g = panel.getGraphics();

        // inner red fill
        g.setColor(Color.RED);
        g.fillRect(20, 10, 100, 50);

        // black outline
        g.setColor(Color.BLACK);
        g.drawRect(20, 10, 100, 50);
    }
}
```
Superimposing shapes

- When ≥ 2 shapes occupy the same pixels, the last drawn "wins."

```java
import java.awt.Graphics;
import java.awt.Color;

public class Car {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();

        g.setColor(Color.BLACK);
        g.fillRect(10, 30, 100, 50);

        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);
        g.fillOval(80, 70, 20, 20);

        g.setColor(Color.CYAN);
        g.fillRect(80, 40, 30, 20);
    }
}
```
Drawing with loops

- The $x,y,w,h$ expressions can use the loop counter variable:

```java
panel.setBackground(Color.YELLOW);
g.setColor(Color.RED);
for (int i = 1; i <= 10; i++) {
    //                 x      y       w   h
    g.fillOval(100 + 20 * i, 5 + 20 * i, 50, 50);
}
```

- Nested loops can be used with graphics:

```java
g.setColor(Color.BLUE);
for (int x = 1; x <= 4; x++) {
    for (int y = 1; y <= 9; y++) {
        g.drawString("Java", x * 40, y * 25);
    }
}
```
Write a method that draws straight lines to create a shape with a curved appearance.

Specify the x and y location of the upper left corner of the drawing and the size of the square to contain the drawing.
Zero-based loops

- Beginning at 0 and using < can make calculating coordinates easier.

```java
DrawingPanel panel = new DrawingPanel(150, 140);
Graphics g = panel.getGraphics();

// horizontal line of 5 20x20 rectangles starting
// at (11, 18); x increases by 20 each time
for (int i = 0; i < 5; i++) {
    g.drawRect(11 + 20 * i, 18, 20, 20);
}
```

- Exercise: Write a variation of the above program that draws the output at right.
  - The bottom-left rectangle is at (11, 98).

```java
for (int i = 0; i < 5; i++) {
    g.drawRect(11 + 20 * i, 98 - 20 * i, 20, 20);
}