Topic 11
Simple Graphics

"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

Based on slides for Building Java Programs by Reges/Stepp, found at http://faculty.washington.edu/stepp/book/

DrawingPanel

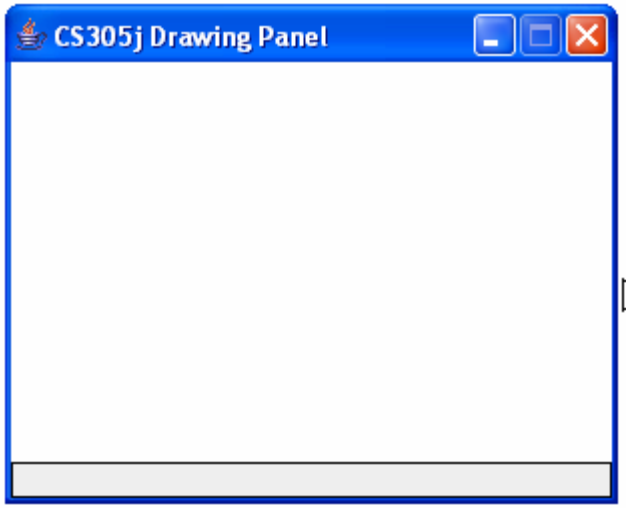
- To make a window appear on the screen, we must create a DrawingPanel object:

```
DrawingPanel <name> = new DrawingPanel(<width>, <height>);
```

  - Example:

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

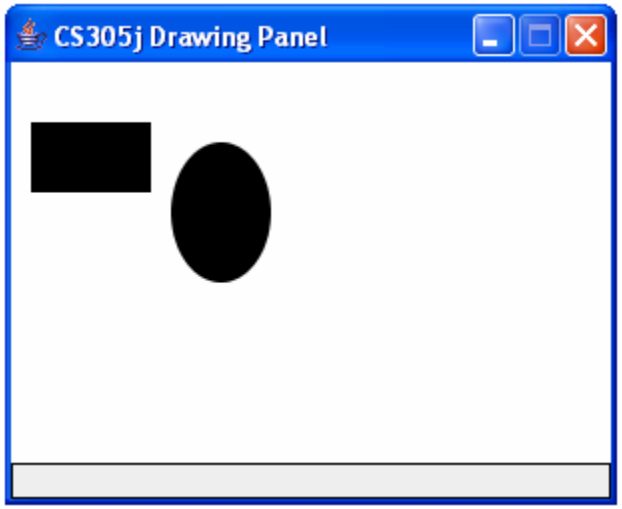
- The window has nothing on it, but we can draw shapes and lines on it using another object of a type named `Graphics`.
  - Using Graphics requires us to place an *import statement* in our program: `import java.awt.*;`



Graphics object

- Shapes are drawn on a DrawingPanel using an object named Graphics.
  - To create a Graphics object for drawing:
    `Graphics` ***<name>*** `=` ***<name>*** `.getGraphics();`
  - Example:
    ```
    Graphics g = panel.getGraphics();
    ```

- Once you have the Graphics object, you can draw shapes by calling methods on it.
  - Example:
    ```
    g.fillRect(10, 30, 60, 35);
    g.fillOval(80, 40, 50, 70);
    ```



Graphics methods

- Here are the drawing commands we can execute:

Method name	Description
drawLine(*x1*, *y1*, *x2*, *y2*)	line between points (*x1*, *y1*), (*x2*, *y2*)
drawOval(*x*, *y*, *width*, *height*)	outline of largest oval that fits in a box of size *width* * *height* with top-left corner at (*x*, *y*)
drawRect(*x*, *y*, *width*, *height*)	outline of rectangle of size *width* * *height* with top-left corner at (*x*, *y*)
drawString(*String*, *x*, *y*)	writes text with bottom-left corner at (x, y)
fillOval(*x*, *y*, *width*, *height*)	entire largest oval that fits in a box of size *width* * *height* with top-left corner at (*x*, *y*)
fillRect(*x*, *y*, *width*, *height*)	entire rectangle of size *width* * *height* with top-left corner at (*x*, *y*)
setColor(*Color*)	Sets Graphics to paint subsequent shapes in the given Color

Calling methods of objects

- Graphics is an "object" that contains methods inside it.
  - When we want to draw something, we don't just write the method's name. We also have to write the name of the Graphics object, which is usually `g`, followed by a dot.
- Calling a method of an object, general syntax:

  <name> **.** ***<method name>*** **(** ***<parameter(s)>*** **)**

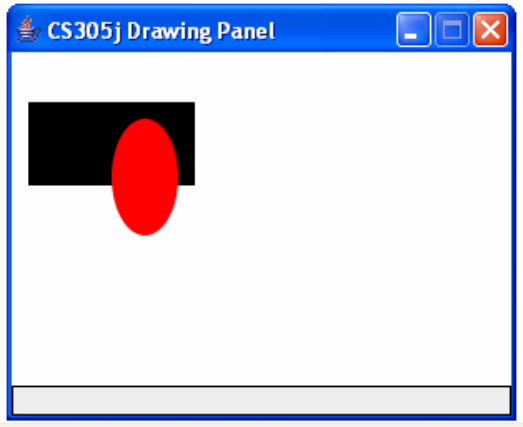
  - Examples:

```
Graphics g = panel.getGraphics();
g.drawLine(20, 30, 90, 10);// tell g to draw a line
```

Colors

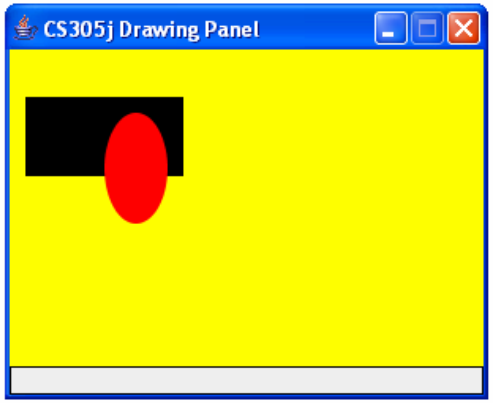
- Shapes can be drawn in many colors.
  - Colors are specified through global constants in the Color class named BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW
  - Example:

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



- The background color of a `DrawingPanel` can be set by calling its `setBackground` method:
  - Example:

```
panel.setBackground(Color.YELLOW);
```



Coordinate system

- Each (x, y) position on the DrawingPanel is represented by one pixel (one tiny dot) on the screen.

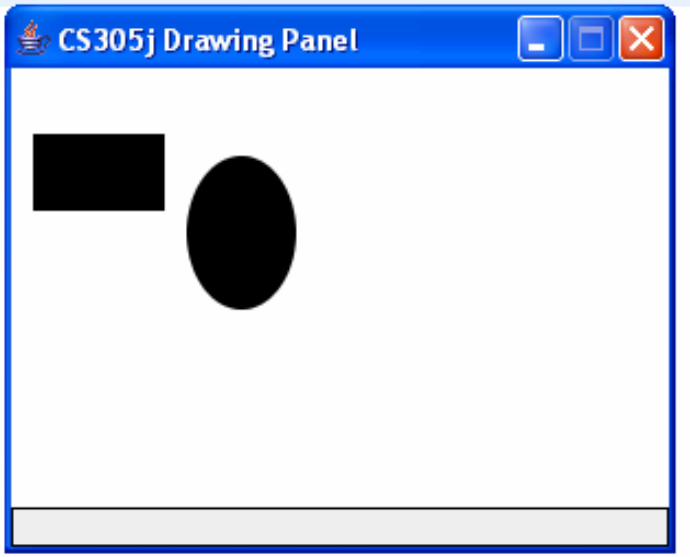
- The coordinate system used by DrawingPanel and Graphics has its origin (0, 0) at the window's top-left corner.
  - The x value increases rightward and the y value increases downward.
  - This is reversed from what you may expect from math classes.

- For example, the rectangle from (0, 0) to (200, 100) looks like this:

```
(0, 0)  +-------+
        |       |
        |       |
        +-------+ (200, 100)
```

Drawing example 1

```
import java.awt.*;

public class DrawingExample1 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(300, 200);

        Graphics g = panel.getGraphics();
        g.fillRect(10, 30, 60, 35);
        g.fillOval(80, 40, 50, 70);
    }
}
```

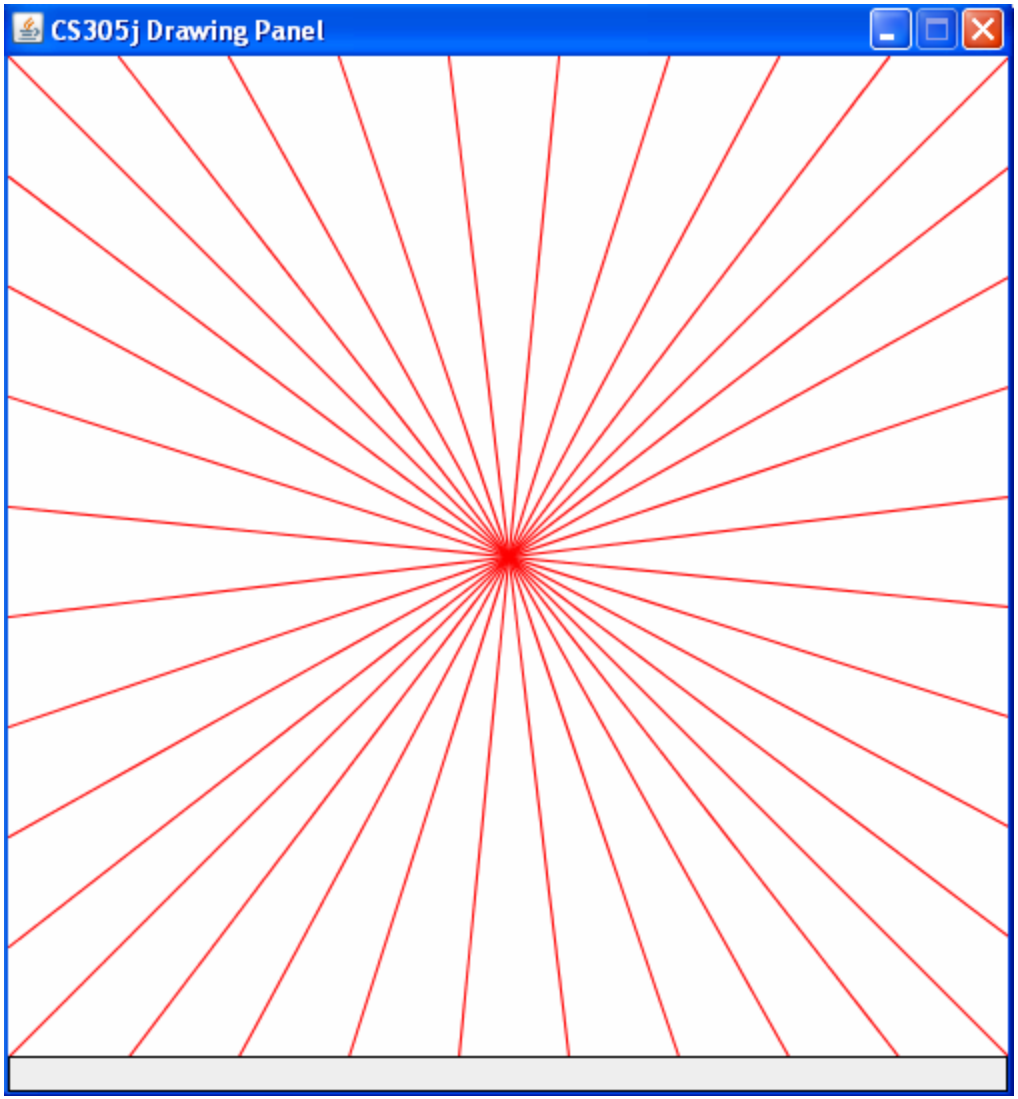


Complicated(??) example

Write a Java program to produce the star burst pattern.

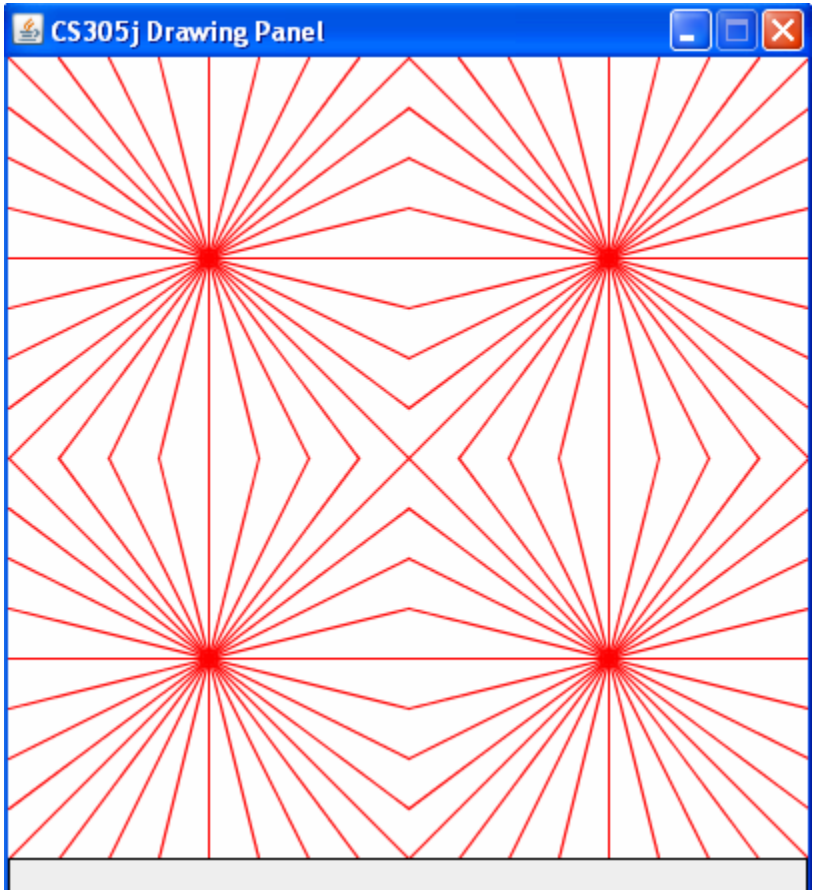
Hard? All lines are the same number of pixels apart at the edges of the panel.

Make in general?



If it is general

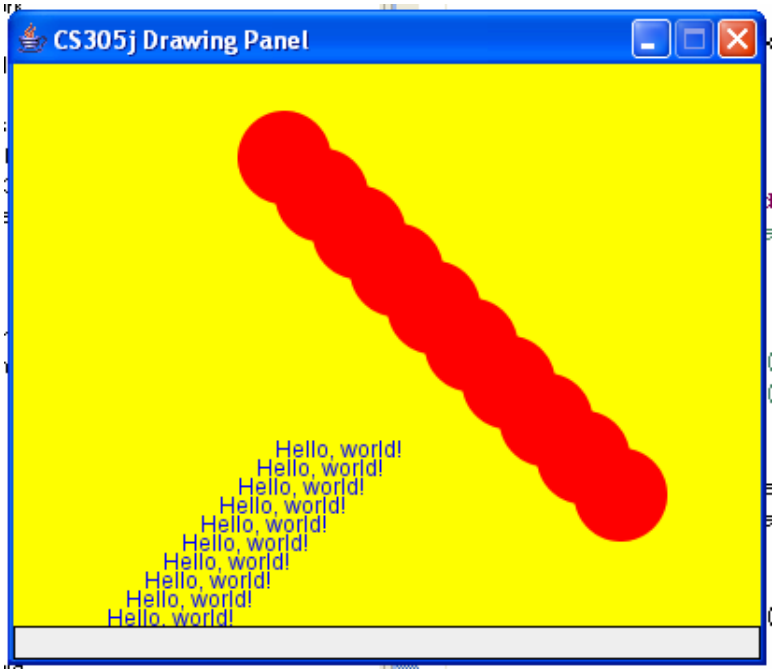
- If we make a method to do the star burst how hard would it be to go to this?



More Examples

- Using for loops, we can draw many repetitions of the same item by varying its x and y coordinates.
  - The x or y coordinate's expression should contain the loop counter, i, so that in each pass of the loop, when i changes, so does x or y.

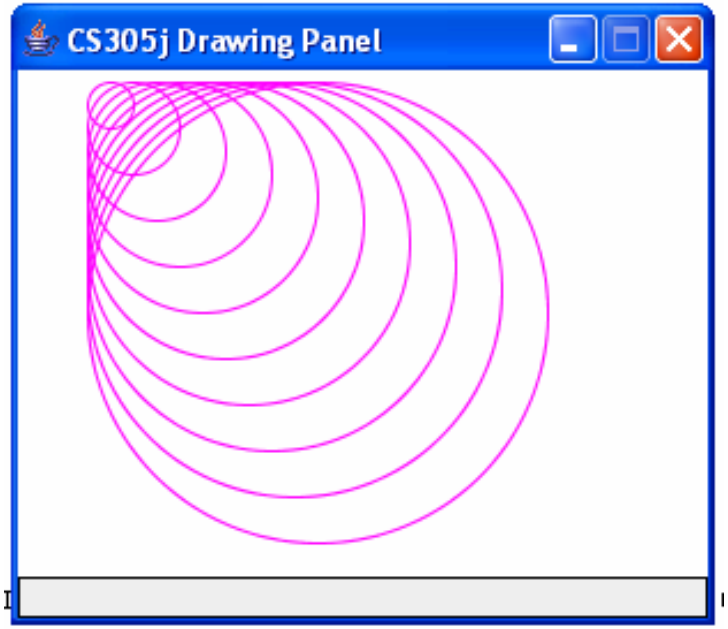
```
DrawingPanel panel = new DrawingPanel(400, 300);
panel.setBackground(Color.YELLOW);

Graphics g = panel.getGraphics();
g.setColor(Color.BLUE);

for (int i = 1; i <= 10; i++) {
    g.drawString("Hello, world!",
          150 - 10 * i, 200 + 10 * i);
}

g.setColor(Color.RED);
for (int i = 1; i <= 10; i++) {
    g.fillOval(100 + 20 * i,
               5 + 20 * i, 50, 50);
}
```



Loops that change size

- A for loop can also vary the size of the shape or figure that it draws.

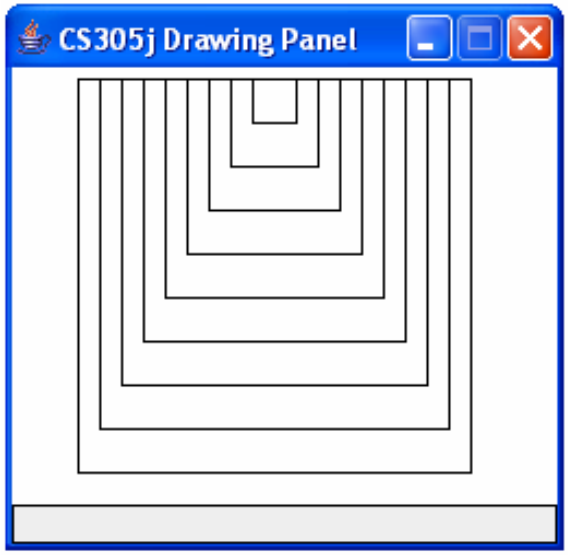
```
DrawingPanel panel = new DrawingPanel(300, 220);

Graphics g = panel.getGraphics();
g.setColor(Color.MAGENTA);
for (int i = 1; i <= 10; i++) {
    g.drawOval(30, 5,
       20 * i, 20 * i);
}
```



A loop that varies both

- The loop in this program affects both the size and shape of the figures being drawn.
  - Each pass of the loop, the square drawn becomes 20 pixels smaller in size, and shifts 10 pixels to the right.

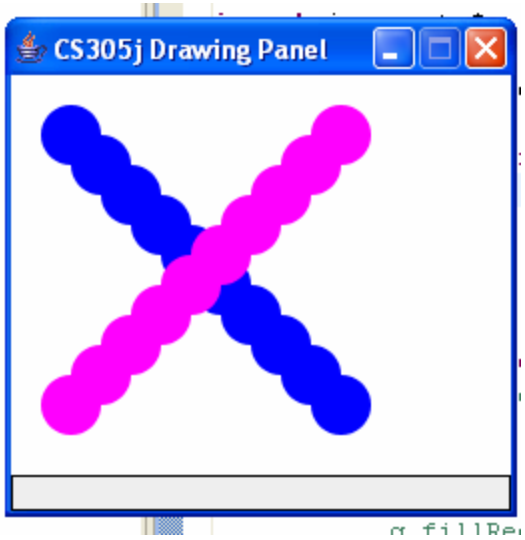
```
DrawingPanel panel = new DrawingPanel(250, 200);

Graphics g = panel.getGraphics();
for (int i = 1; i <= 10; i++) {
    g.drawRect(20 + 10 * i, 5,
       200 - 20 * i, 200 - 20 * i);
}
```



Drawing example 2

- What sort of figure does the following code draw?

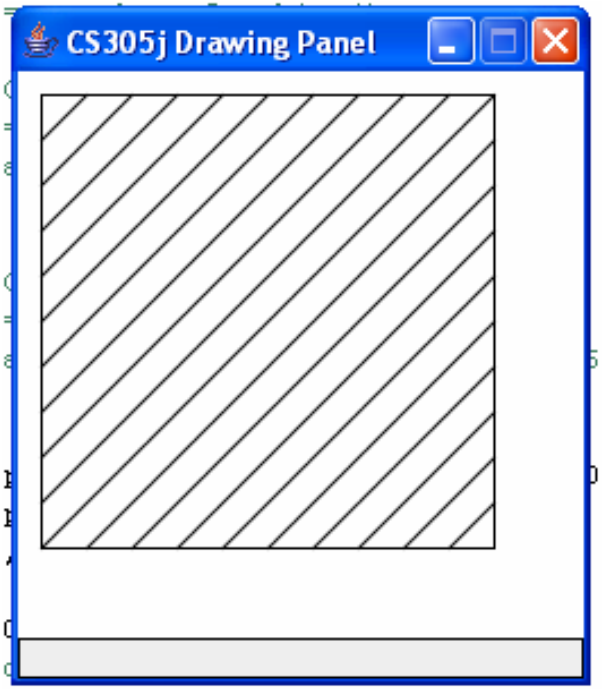
```
import java.awt.*;

public class DrawingExample2 {
   public static final int NUM_CIRCLES = 10;

   public static void main(String[] args) {
      DrawingPanel panel = new DrawingPanel(250, 200);
      Graphics g = panel.getGraphics();

      g.setColor(Color.BLUE);
      for (int i = 1; i <= NUM_CIRCLES; i++) {
         g.fillOval(15 * i, 15 * i, 30, 30);
      }

      g.setColor(Color.MAGENTA);
      for (int i = 1; i <= NUM_CIRCLES; i++) {
         g.fillOval(15 * (NUM_CIRCLES + 1 - i), 15 * i, 30, 30);
      }
   }
}
```



Loops that begin at 0

- Often when working with graphics (and with later loops in general), we begin our loop count at 0 and end one repetition earlier.
  - A loop that repeats from 0 to < 10 still repeats 10 times, just like a loop that repeats from 1 to <= 10.
  - But when the loop counter variable i is used to set the figure's coordinates, often starting i at 0 gives us the coordinates we want.

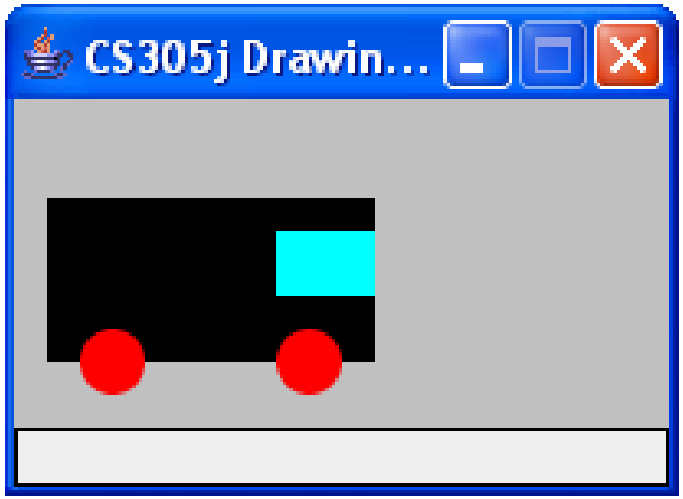
```
DrawingPanel panel = new DrawingPanel(250, 250);
Graphics g = panel.getGraphics();
g.drawRect(10, 10, 200, 200);

for (int i = 0; i < 10; i++) {
    // lines on the upper-left half
    g.drawLine(10, 10 + 20 * i,
               10 + 20 * i, 10);

    // lines on the lower-right half
    g.drawLine(10 + 20 * i, 210,
               210, 10 + 20 * i);
}
```



Superimposing shapes

- Drawing one shape on top of another causes the last shape to appear on top of the previous one(s).

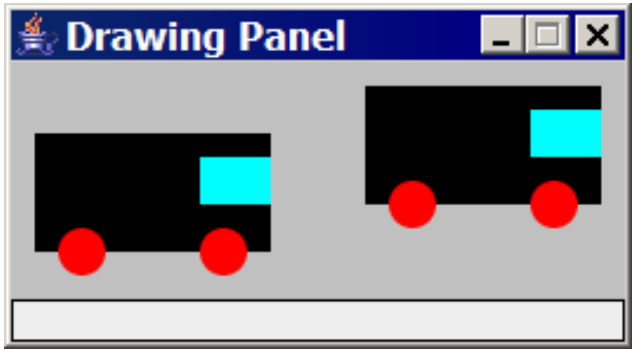
```
import java.awt.*;

public class DrawingExample3 {
    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 parameters

- Imagine that we want to draw two figures as shown in the picture below.
- If you wish to repeat the same figure multiple times on the drawing panel, write a method that draws that figure and accepts the x/y position as parameters.
  - Adjust all of your x/y coordinates of your drawing commands to take into account the parameters.
  - Since you'll need to send commands to the Graphics g in order to draw the now parameterized figure, you should also pass Graphics g as a parameter.

```
public static void drawCar(Graphics g, int x, int y) {
    g.setColor(Color.BLACK);
    g.fillRect(x, y, 100, 50);

    // ...
}
```

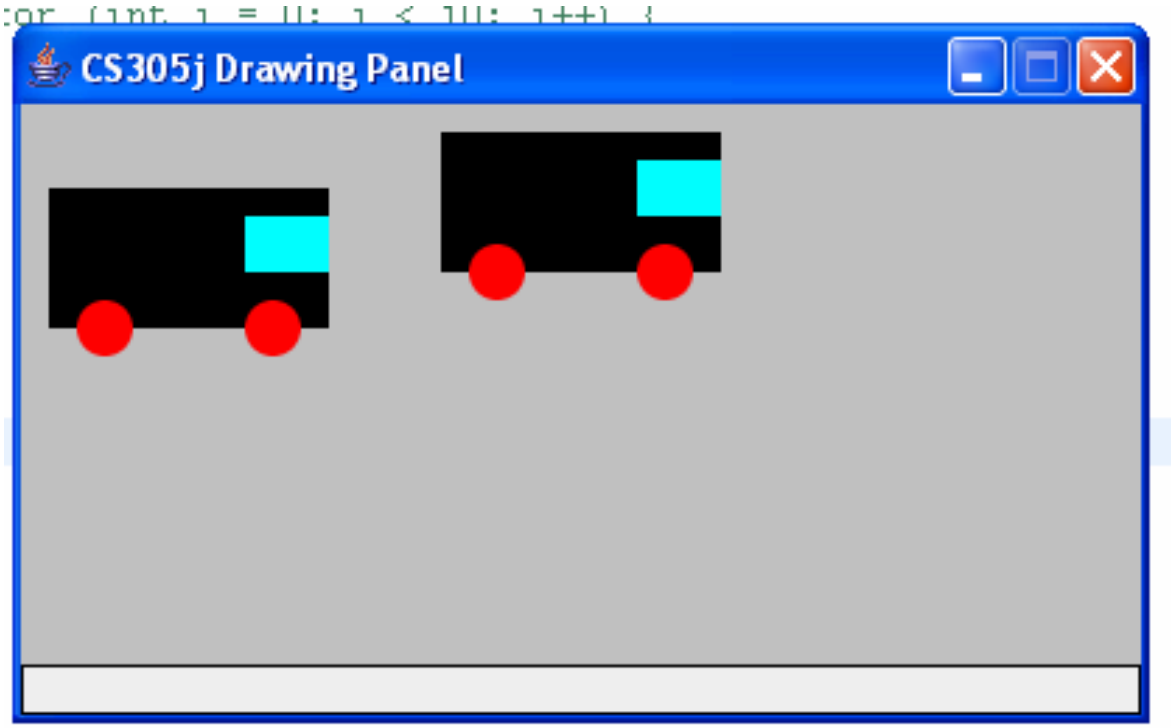


Drawing with parameters

- Here is the complete program that uses a parameterized method to draw multiple car figures:

```
import java.awt.*;
public class DrawingWithParameters {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(260, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30);
        drawCar(g, 150, 10);
    }

    public static void drawCar(Graphics g, int x, int y) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, 100, 50);

        g.setColor(Color.RED);
        g.fillOval(x + 10, y + 40, 20, 20);
        g.fillOval(x + 70, y + 40, 20, 20);

        g.setColor(Color.CYAN);
        g.fillRect(x + 70, y + 10, 30, 20);
    }
}
```

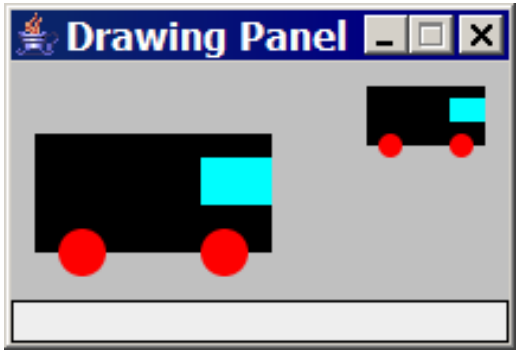
Result



More parameters

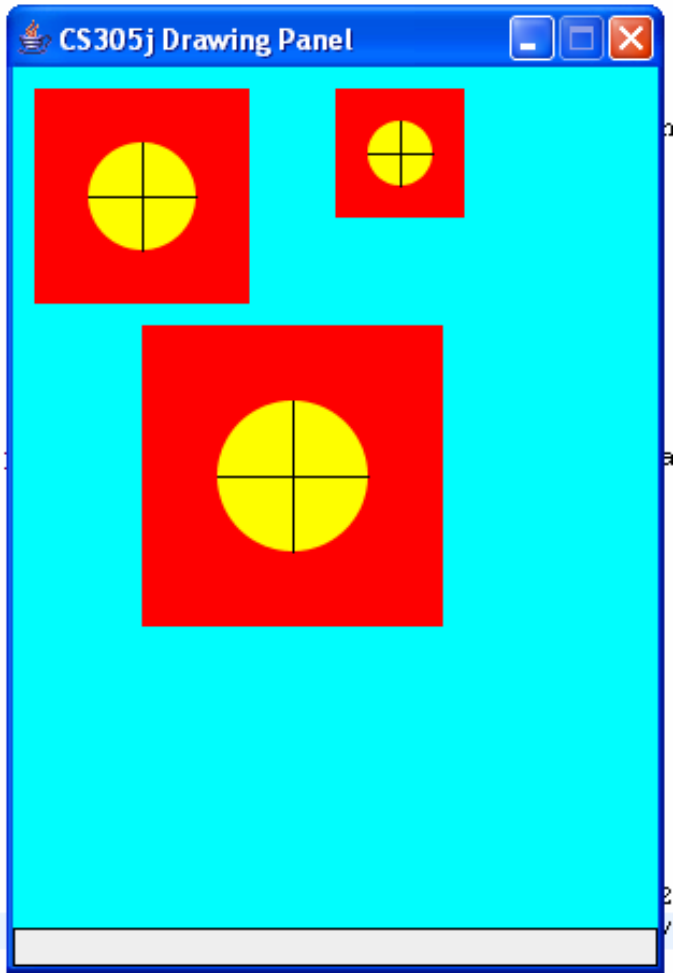
- A new version where the cars can be resized:

```
public class DrawingWithParameters2 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(210, 100);
        panel.setBackground(Color.LIGHT_GRAY);

        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30, 100);
        drawCar(g, 150, 10, 50);
    }

    public static void drawCar(Graphics g, int x, int y, int size) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, size, size / 2);

        g.setColor(Color.RED);
        g.fillOval(x + size / 10, y + 2 * size / 5,
                   size / 5, size / 5);
        g.fillOval(x + 7 * size / 10, y + 2 * size / 5,
                   size / 5, size / 5);

        g.setColor(Color.CYAN);
        g.fillRect(x + 7 * size / 10, y + size / 10,
                   3 * size / 10, size / 5);
    }
}
```



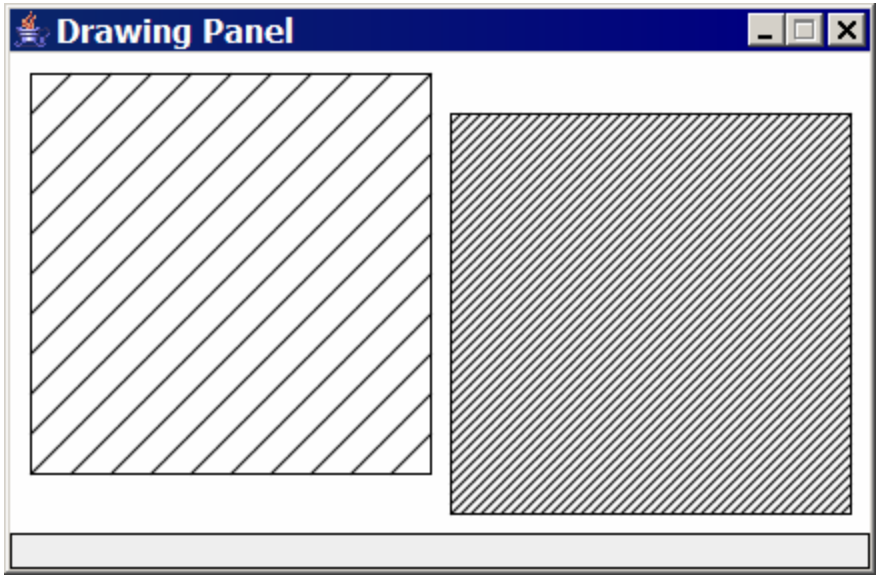
Parameterized figure exercise

- Let's write a program together that will display the following figures on a drawing panel of size 300x400:
  - top-left figure:
    - overall size = 100
    - top-left corner = (10, 10)
    - oval size = 50
    - inner top-left corner = (35, 35)
  - top-right figure:
    - overall size = 60
    - top-left corner = (150, 10)
    - oval size = 30
    - inner top-left corner = (165, 25)
  - bottom figure:
    - overall size = 140
    - top-left corner = (60, 120)
    - oval size = 70
    - inner top-left corner = (95, 155)



Parameterized figure exercise

- Write a program that will display the following figure using parameterized methods.
  - Start with the "loops that begin at 0" program shown earlier in the slides.
  - Use a parameter for the number of lines (as well as any other parameters you need).
  - The second square is still 200x200 in size, but it is at (220, 30) and has 40 line loops compared to the original figure's 10.



Animation with sleep

- The `DrawingPanel` has a method named `sleep` that makes your program pause for a given number of milliseconds (thousandths of a second).

- You can use the `sleep` method to produce simple animations.

```
DrawingPanel panel = new DrawingPanel(250, 200);
Graphics g = panel.getGraphics();

g.setColor(Color.BLUE);
for (int i = 1; i <= NUM_CIRCLES; i++) {
    g.fillOval(15 * i, 15 * i, 30, 30);
    panel.sleep(500);
}
```

  - Try adding sleep commands to loops in past exercises in this chapter and watch the panel draw itself piece by piece!