

A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar. The wall is partially visible, extending from the left edge towards the center of the frame.

Building Java Programs

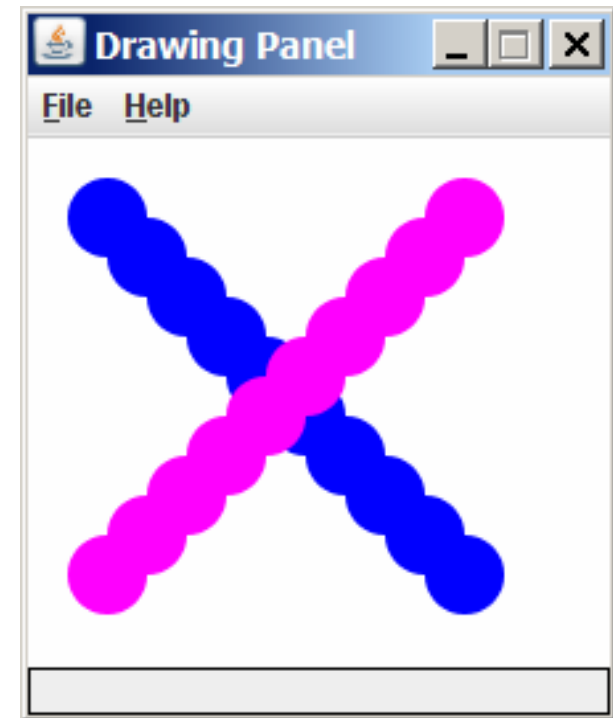
Supplement 3G: Graphics

Lecture outline

- drawing 2D graphics
 - DrawingPanel and Graphics objects
 - drawing and filling shapes
 - coordinate system
 - colors
 - drawing with loops
 - drawing with parameterized methods
 - basic animation

Graphical objects

- We will draw graphics using these kinds of *objects*:
 - `DrawingPanel`: A window on the screen.
 - This is not part of Java; it is provided by the authors.
 - `Graphics`: A "pen" that can draw shapes/lines onto a window.
 - `Color`: The colors that indicate what color to draw our shapes.
- **object**: An entity that contains data and behavior.
 - data: Variables inside the object.
 - behavior: Methods inside the object.



DrawingPanel

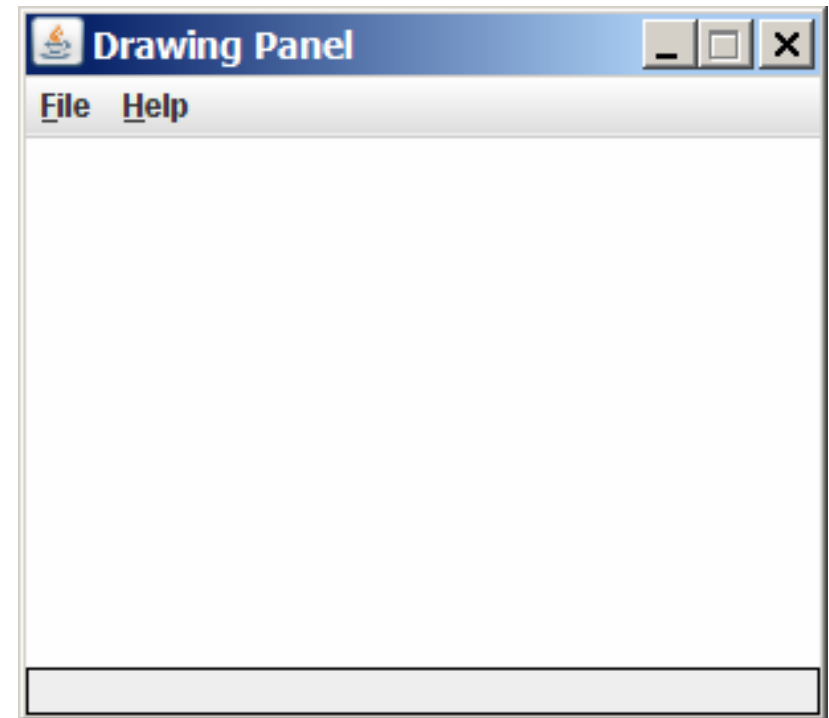
- To create a window, construct a DrawingPanel object:

```
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 class named Graphics.



Graphics

- Shapes are drawn using an object of class `Graphics`.
 - You must place an import declaration in your program:

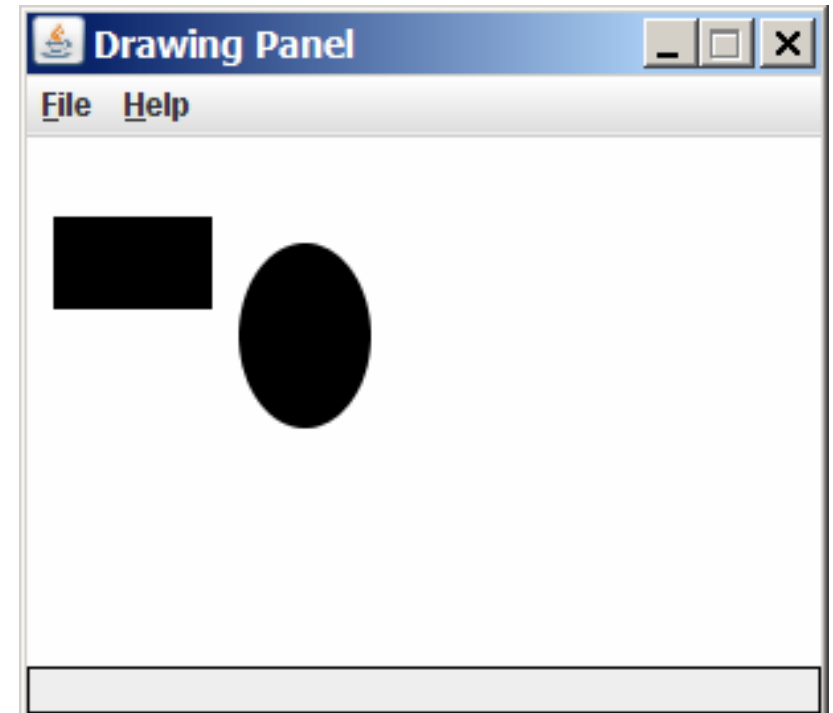
```
import java.awt.*;
```
 - Access it by calling `getGraphics` on your `DrawingPanel`.
 - Example:

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

- Once you have the `Graphics` object, draw shapes by calling its methods.

- Example:

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

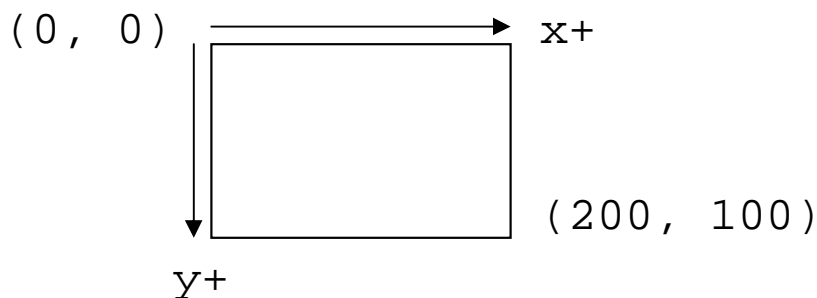


Graphics methods

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

Coordinate system

- Each (x, y) position on the `DrawingPanel` represents a *pixel* (short for "picture element").
- $(0, 0)$ is at the window's top-left corner.
 - x increases rightward and the y increases downward.
(The y is reversed from what you may expect.)
- The rectangle from $(0, 0)$ to $(200, 100)$ looks like this:

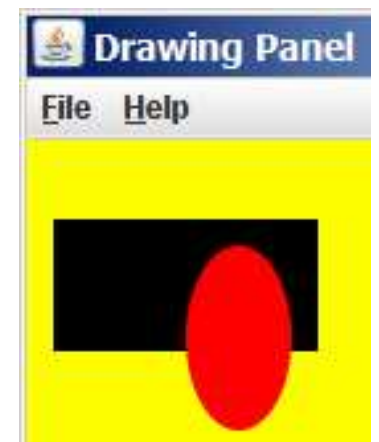
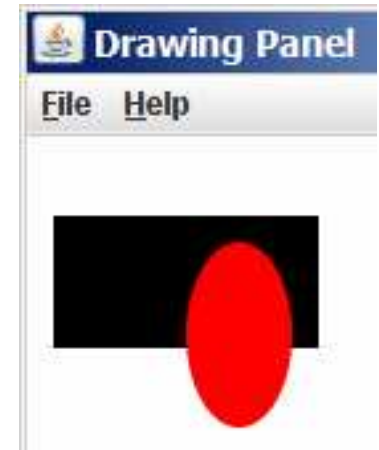


Colors

- Colors are specified by `Color` class constants named: `BLACK`, `BLUE`, `CYAN`, `DARK_GRAY`, `GRAY`, `GREEN`, `LIGHT_GRAY`, `MAGENTA`, `ORANGE`, `PINK`, `RED`, `WHITE`, `YELLOW`
 - Pass these to the `Graphics` object's `setColor` method.
 - Example:

```
g.setColor(Color.BLACK);  
g.fillRect(10, 30, 100, 50);  
g.setColor(Color.RED);  
g.fillOval(60, 40, 40, 70);
```
- The background color can be set by calling `setBackground` on the `DrawingPanel`:
 - Example:

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



Outlined shapes

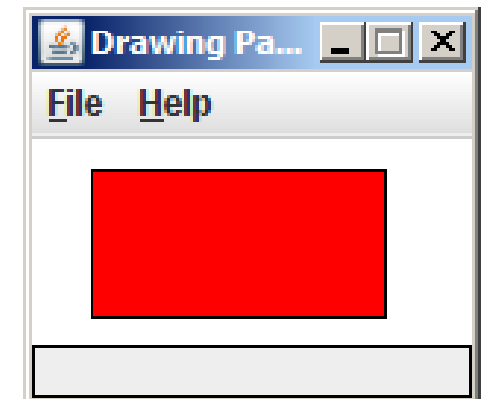
- To draw a shape filled in one color and outlined in another, first *fill* it in the fill color and then *draw* the same shape with its outline color.

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

public class DrawOutline {
    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

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

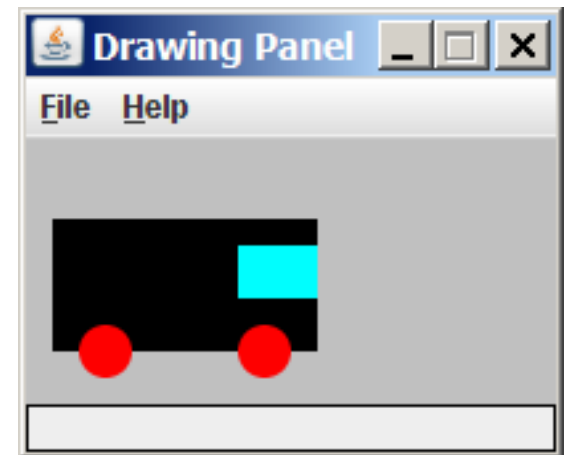
```
import java.awt.*;

public class DrawCar {
    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);
    }
}
```

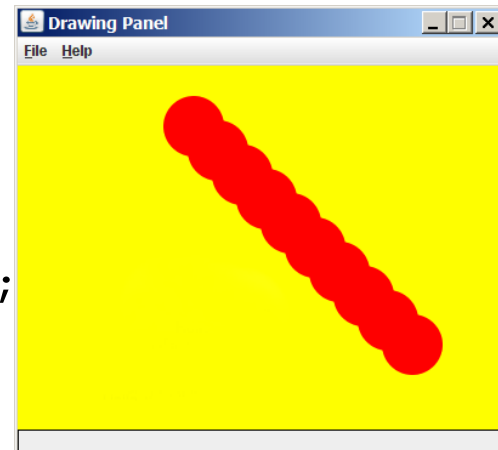


Drawing with loops

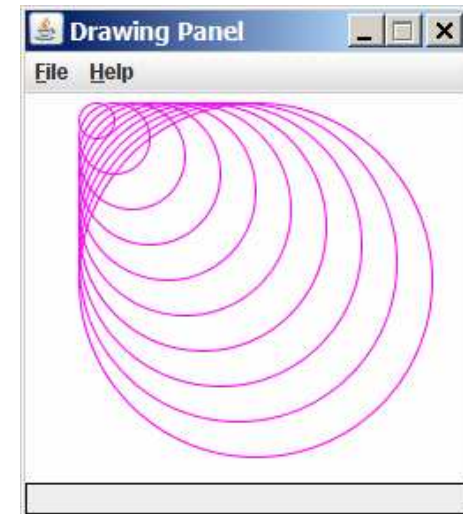
- Draw many shapes at different x/y with `for` loops.
 - The `x, y, w, h` expression can contain the loop counter, `i`.

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

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



```
DrawingPanel panel = new DrawingPanel(250, 220);
Graphics g = panel.getGraphics();
g.setColor(Color.MAGENTA);
for (int i = 1; i <= 10; i++) {
    g.drawOval(30, 5, 20 * i, 20 * i);
}
```

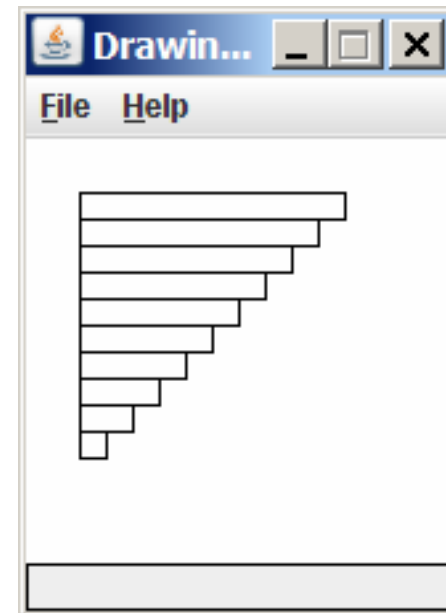


Loops that begin at 0

- It can be easier to begin our loop at 0 and use `<`.
 - A loop from 0 to `< 10` still repeats 10 times, just like 1 to `<= 10`.
 - Starting at 0 sometimes makes coordinates easier to compute.
- Example:
 - Draw ten stacked rectangles starting at (20, 20), height 10, width starting at 100 and decreasing by 10 each time:

```
DrawingPanel panel = new DrawingPanel(160, 160);  
Graphics g = panel.getGraphics();
```

```
for (int i = 0; i < 10; i++) {  
    g.drawRect(20, 20 + 10 * i,  
               100 - 10 * i, 10);  
}
```

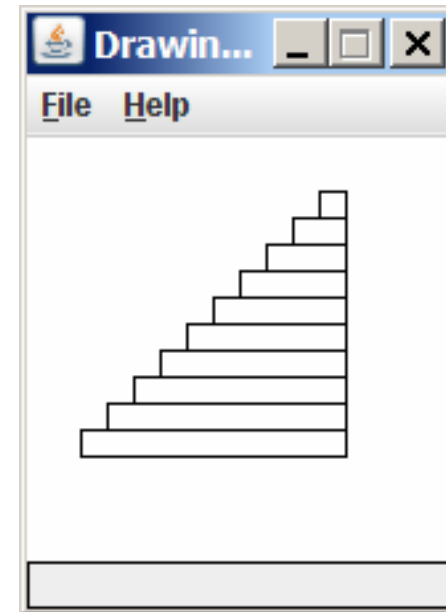
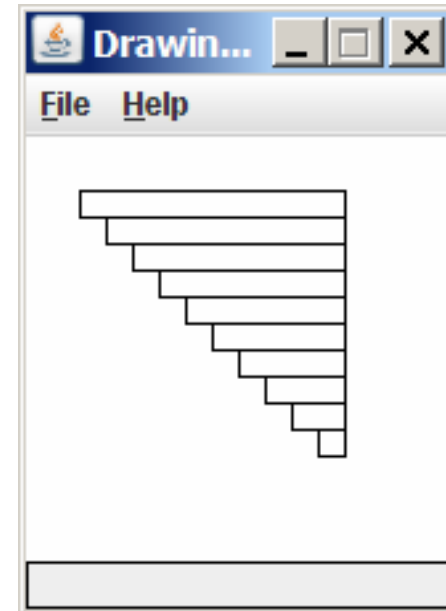


Drawing w/ loops questions

- Code from previous slide:

```
DrawingPanel panel = new DrawingPanel(160, 160);  
Graphics g = panel.getGraphics();  
  
for (int i = 0; i < 10; i++) {  
    g.drawRect(20, 20 + 10 * i,  
              100 - 10 * i, 10);  
}
```

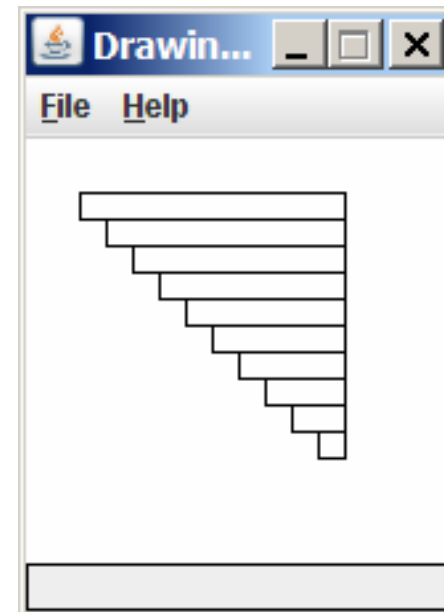
- Write variations of the preceding program that draw the figures at right as output.



Drawing w/ loops answers

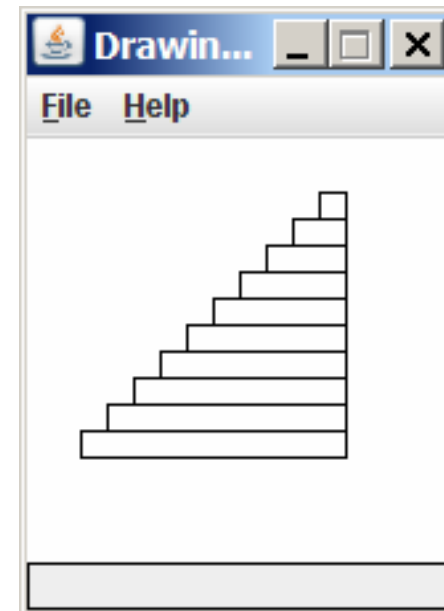
■ Solution #1:

```
Graphics g = panel.getGraphics();  
for (int i = 0; i < 10; i++) {  
    g.drawRect(20 + 10 * i, 20 + 10 * i,  
               100 - 10 * i, 10);  
}
```



■ Solution #2:

```
Graphics g = panel.getGraphics();  
for (int i = 0; i < 10; i++) {  
    g.drawRect(110 - 10 * i, 20 + 10 * i,  
               10 + 10 * i, 10);  
}
```



Drawing with methods

- It is possible to draw graphics in multiple methods.
 - You must pass `Graphics g` as a parameter.

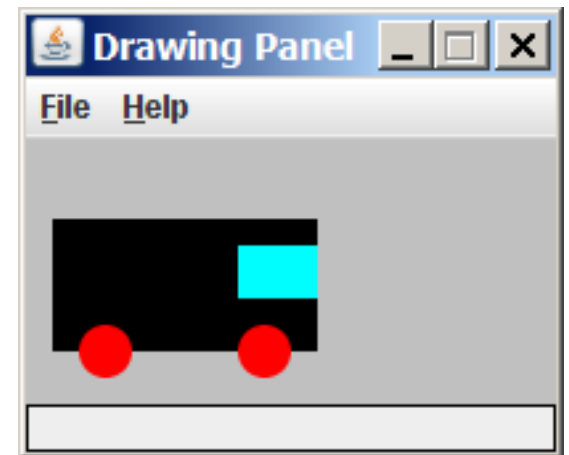
```
import java.awt.*;

public class DrawCar1 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        drawCar(g);
    }

    public static void drawCar(Graphics g) {
        g.setColor(Color.BLACK);
        g.fillRect(10, 30, 100, 50);

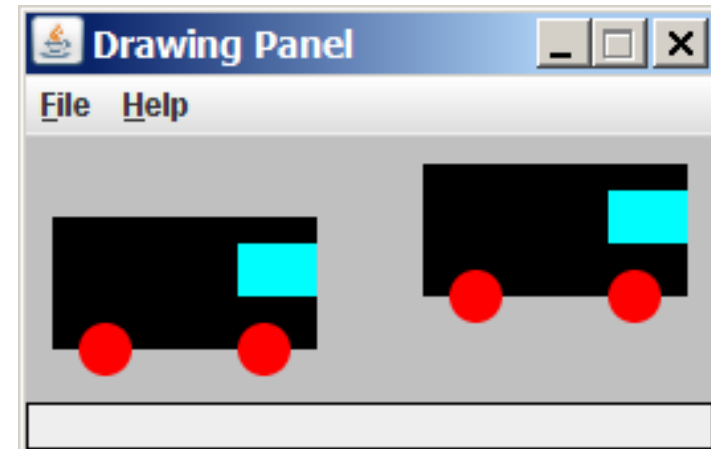
        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);
        g.fillOval(80, 70, 20, 20);

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



Parameterized figures

- To draw the same figure many times, write a method that accepts the x/y position as parameters.
 - Adjust your drawing commands to use the parameters.
- Modify the previous car-drawing method to work at any location, so that it can produce the following image.
 - One car's top-left corner is at (10, 30).
 - The other car's top-left corner is at (150, 10).



Drawing parameters answer

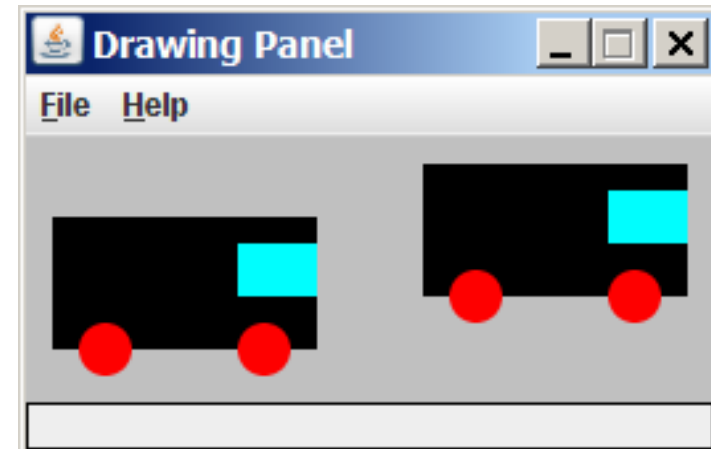
```
import java.awt.*;

public class DrawCar2 {
    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);

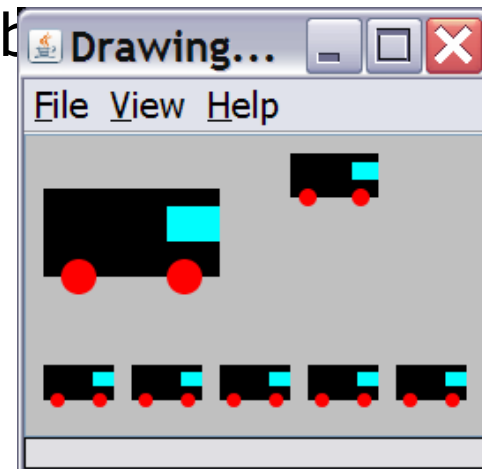
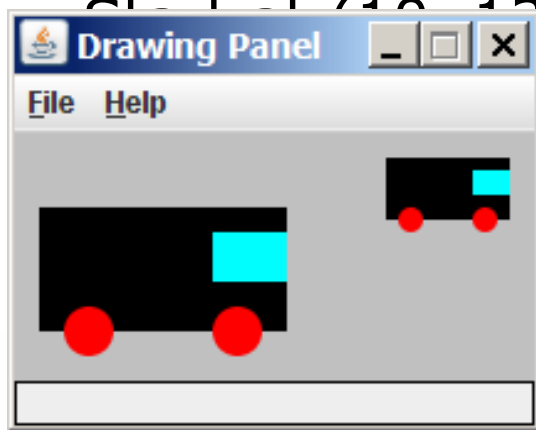
        g.setColor(Color.RED);
        g.fillOval(x + 10, y + 40, 20, 20);
        g.fillOval(x + 70, y + 40, 20, 20);

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



Drawing parameter question

- Methods can accept any number of parameters to adjust the figure's appearance.
- Exercise:
Write a new `drawCar` that allows the cars to be drawn at any size, like the picture at left.
 - Existing car: size 100. Second car: (150, 10), size 50.
- Once you have this working, use a `for` loop with your method to draw a line of cars, like the picture at right.



Drawing parameter solution

```
import java.awt.*;

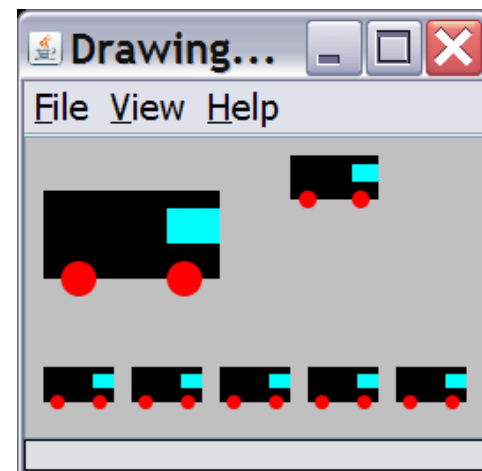
public class DrawCar3 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(210, 100);
        panel.setBackground(Color.LIGHT_GRAY);

        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30, 100);
        drawCar(g, 150, 10, 50);
        for (int i = 0; i < 5; i++) {
            drawCar(g, 10 + i * 50, 130, 40);
        }
    }

    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);
    }
}
```



Custom colors

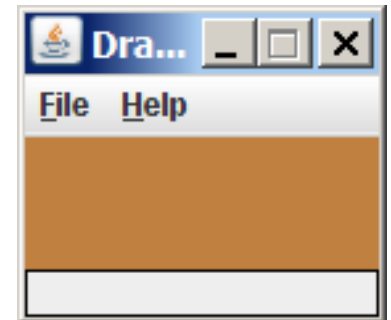
- It is also legal to construct a `Color` object of your own.
 - Colors are specified by three numbers (`ints` from 0 to 255) representing the amount of red, green, and blue.
 - Computers use red-green-blue or "RGB" as primary colors.

- Example:

```
DrawingPanel panel = new DrawingPanel(80, 50);
```

```
Color brown = new Color(192, 128, 64);
```

```
panel.setBackground(brown);
```



- or:

```
DrawingPanel panel = new DrawingPanel(80, 50);
```

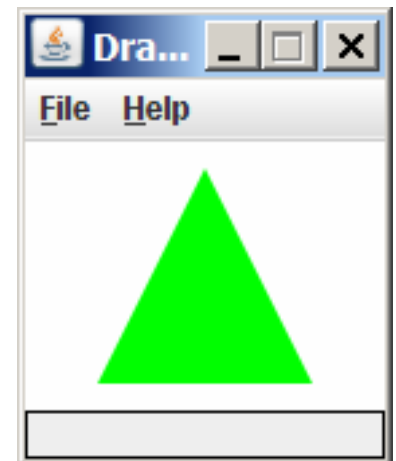
```
panel.setBackground(new Color(192, 128, 64));
```

Drawing polygons

- Polygon objects represent arbitrary shapes.
 - Add points to a Polygon using its `addPoint(x, y)` method.

- **Example:**

```
DrawingPanel p = new DrawingPanel(100, 100);  
Graphics g = p.getGraphics();  
Polygon poly = new Polygon();  
poly.addPoint(10, 90);  
poly.addPoint(50, 10);  
poly.addPoint(90, 90);  
g.setColor(Color.GREEN);  
g.fillPolygon(poly);
```



Animation with sleep

- `DrawingPanel` has a method named `sleep` that pauses your program for a given number of milliseconds.

- You can use `sleep` 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.