Building Java Programs

Graphics

reading: Supplement 3G

videos: Ch. 3G #1-2

Code "Libraries"

- For homework 3 (and others), we'll use a Graphics Library written by the textbook authors
  - Library: Code written to make it easier to write many programs
  - Graphics library: Provide features like "draw a black oval"
    - Library takes care of all the drawing details
    - Library is useful for lots of different programs
    - "Feels like" Java has all these features, but it's really just methods and *objects* defined by the library

- Steps for using a library
  1. What "set up" do I have to do to use the library?
  2. What are the basic features of the library?
  3. What are the patterns for making the features useful?

1. Set-up

- The library is in `DrawingPanel.java`
  - On course web-site
  - Must be in the same directory as your program

- Your program must have

```
import java.awt.*;
```

  in your file *before* public class …
  - Otherwise "things" the library gives you won't be defined and your program won't compile
  - These "things" are kinds of objects (classes) defined in Java's "package" called java.awt
  - import says you want these things to be visible to your program

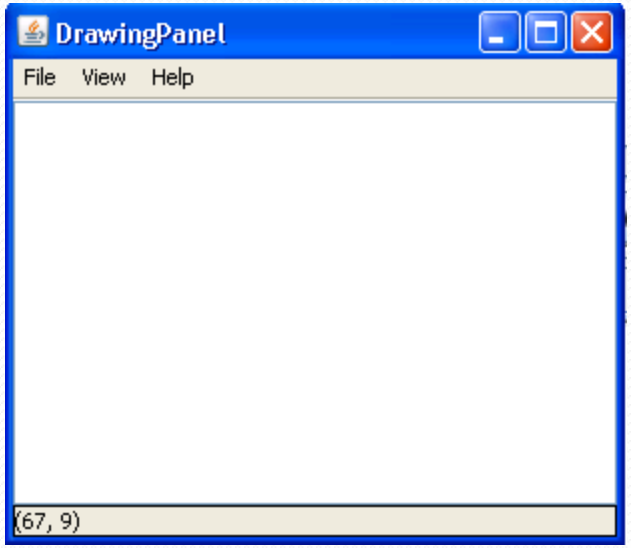
2. Basics, part 1

A complete program using the library to draw 2 ovals:

```
import java.awt.*;
public class MyFirstDrawing {
  public static void main(String[] args) {
    DrawingPanel panel = new DrawingPanel(300,200);
    Graphics g = panel.getGraphics();
    g.setColor(Color.BLACK);
    g.drawOval(100,100,50,50);
    g.drawOval(125,100,75,75);
  }
}
```

Line by line

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

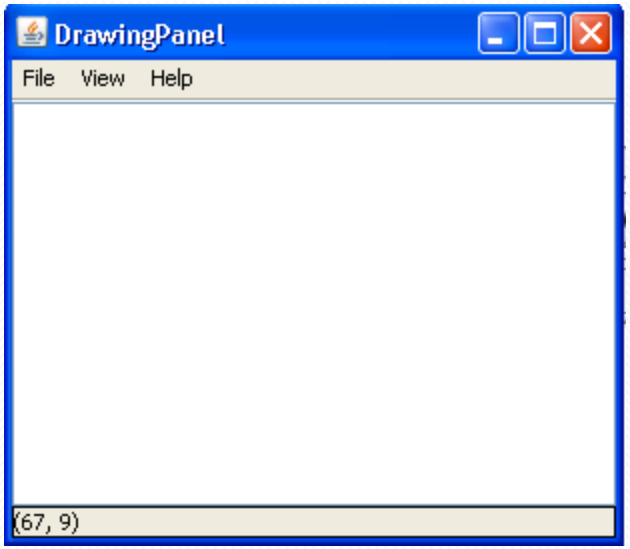


1. Create a new DrawingPanel
   - A canvas to draw things on
   - Make it 300 *pixels* wide and 200 *pixels* high
     - These are parameters to the DrawingPanel *constructor*

2. Store a reference to this new thing in a variable `panel`
   - The library made `DrawingPanel` a type (like `int`, `String`)
   - Otherwise nothing novel about this part, just declaration and initialization

Line by line

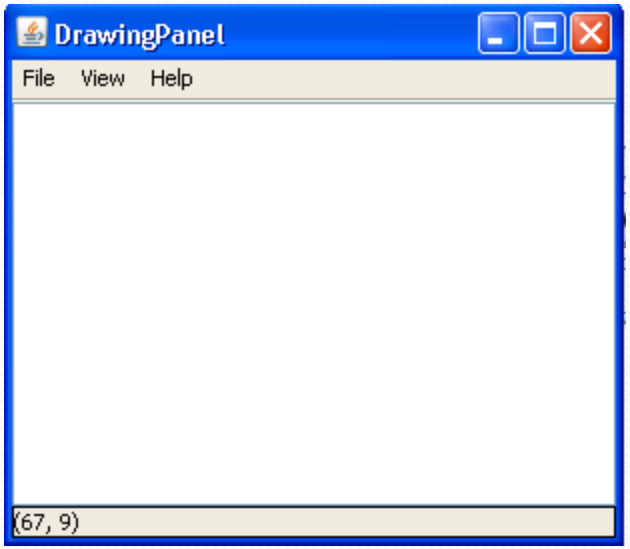
```
DrawingPanel panel = new DrawingPanel(300,200);
Graphics g = panel.getGraphics();
```



1. Call the panel's getGraphics method
   - *Returns* a Graphics object (a "pen" for drawing with on the canvas)
   - (We'll learn how to write methods that return things next time)
2. Store a reference to this "pen" in a variable `g`
   - Again, this part is old news, the new things are:
     - A DrawingPanel has methods we can call
       - They are part of a DrawingPanel
     - These methods can return things

Line by line

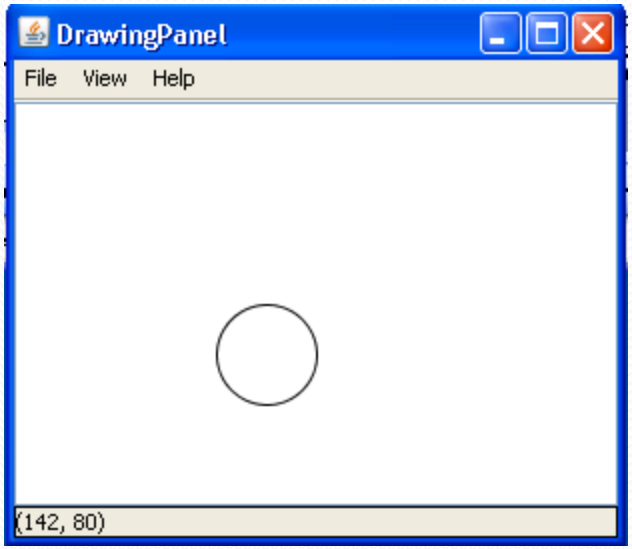
```
DrawingPanel panel = new DrawingPanel(300,200);
Graphics g = panel.getGraphics();
g.setColor(Color.BLACK);
```



- A `Graphics` object, like the one in `g`, also has methods we can call
- The `setColor` method takes a parameter, which is a Color
- The `setColor` method changes the pen's color
  - No immediate effect; affects subsequent drawings

Line by line

```
DrawingPanel panel = new DrawingPanel(300,200);
Graphics g = panel.getGraphics();
g.setColor(Color.BLACK);
g.drawOval(100,100,50,50);
```

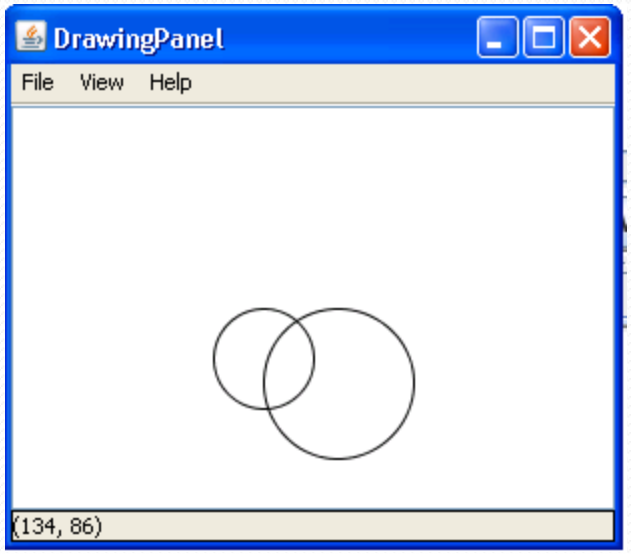


- Another method a Graphics object has draws an oval

- The parameters describe its position and its size
  - See how useful parameters are!
  - Details on which parameter is which a little later
    - (and in book)

Line by line

```
DrawingPanel panel = new DrawingPanel(300,200);
Graphics g = panel.getGraphics();
g.setColor(Color.BLACK);
g.drawOval(100,100,50,50);
g.drawOval(125,100,75,75);
```



- Keep drawing objects to make a picture

- Teaser: Everything we've learned will help us automate picture drawing
  - Example: A loop to draw similar shapes near each other

Where are we

1. What "set up" do I have to do to use the library? *Done*
2. What are the basic features of the library? *In progress*
3. What are the patterns for making the features useful?

Next steps:

- The general organization of the library
- More basic features (rectangles, filling, colors, etc.)
- How this is *object-oriented* and the new Java features we are using

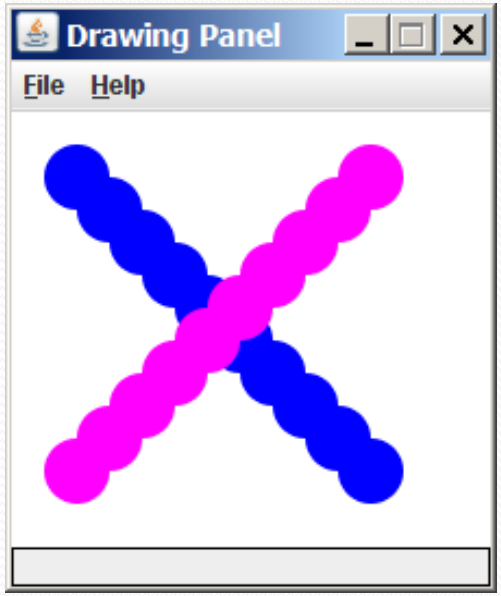
Then the fun stuff:

- Using loops and parameters to make cool and useful pictures

Graphical objects - Recap

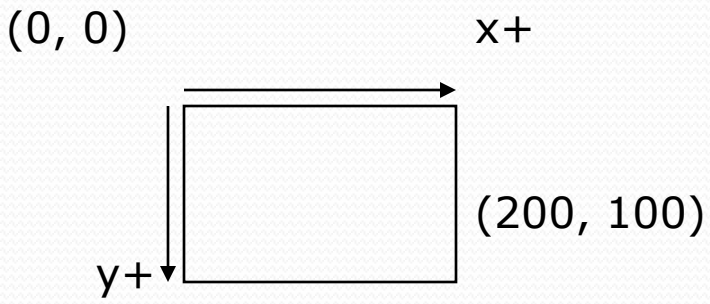
The library gives us 3 new kinds of *objects*:

- `DrawingPanel`: A window on the screen.
- `Graphics`: A "pen" to draw shapes/lines on a window.
- `Color`: Colors in which to draw shapes.



Coordinate system

- Each (x, y) position is a *pixel* ("picture element").
- (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

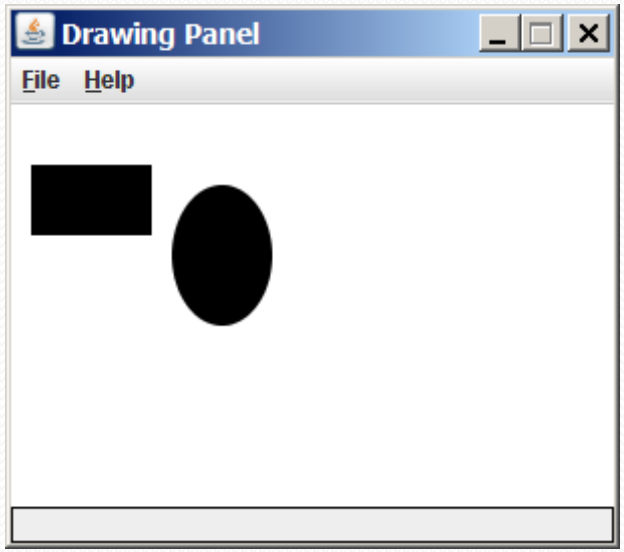
"Pen" objects that can draw lines and shapes

- Access it by calling `getGraphics` on your `DrawingPanel`.

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

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

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



`Graphics` methods

Method name	Description
`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*)
`g.setColor(`**Color**`);`	set `Graphics` to paint any following shapes in the given color

Color

- Create one using Red-Green-Blue (RGB) values from 0-255

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

  - Example:

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

- Or use a predefined `Color` class constant (more common)

```
Color.CONSTANT_NAME
```

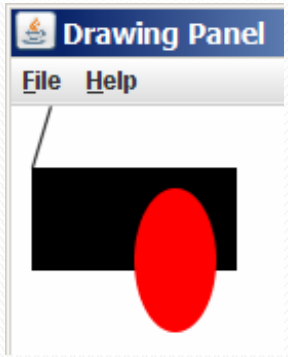
  where **CONSTANT_NAME** is one of:

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

Using Colors

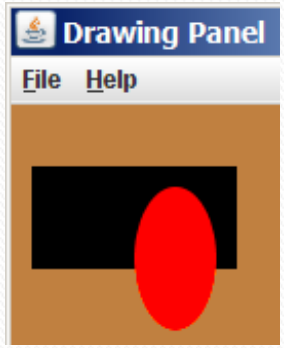
- Pass a `Color` to `Graphics` object's `setColor` method
  - Subsequent shapes will be drawn in the new color.

```
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.

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



Mini-Exercises

- Write a program that draws a solid blue circle with radius 60 centered at x=100, y=100.
- Extend your program to outline the circle in black.
- Reminders:

```
import java.awt.*;

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

`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.fillOval(`**x**, **y**, **width**, **height**`);`	fill largest oval that fits in a box of size *width* * *height* with top-left at (*x*, *y*)
`g.setColor(`**Color**`);`	set `Graphics` to paint any following shapes in the given color

Mini-exercises - solutions

```
import java.awt.*;  // so I can use Graphics
public class CircleExample {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 200);
        Graphics g = panel.getGraphics();
        // to make a blue circle with radius 60,
        // draw an oval with width=height=120
        g.setColor(Color.BLUE);
        g.fillOval(40, 40, 120, 120);
        // black outline
        g.setColor(Color.BLACK);
        g.drawOval(40, 40, 120, 120);
    }
}
```

Objects (briefly)

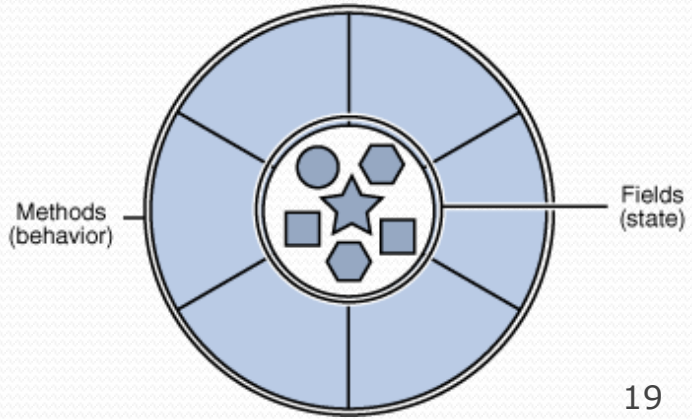
- **object:** An entity that contains data and behavior.
  - *data*: Variables inside the object.
  - *behavior*: Methods inside the object.
    - You interact with the methods; the data is hidden in the object.

- Constructing (creating) an object:

```
type objectName = new type(parameters);
```

- Calling an object's method:

```
objectName.methodName(parameters);
```



Object-oriented

Two perspectives on, for example, setting a pen's color:

1. I have a procedure for changing a pen's color and and I will apply it to this pen I have
   - "Verb-oriented": focus is on the changer you have
   - Would look like setColor(g,Color.BLACK)

2. I consider a pen's color to be part of the pen, so to change the color I will use one of the pen's methods
   - "Noun-oriented": focus is on the pen and what it can do
   - Looks like `g.setColor(Color.BLACK);`

Our Graphics library takes the second approach

- Most Java libraries do because the language has good support for defining and using libraries this way

Where are we

1. What "set up" do I have to do to use the library? *Done*
2. What are the basic features of the library? *Done*
3. What are the patterns for making the features useful?

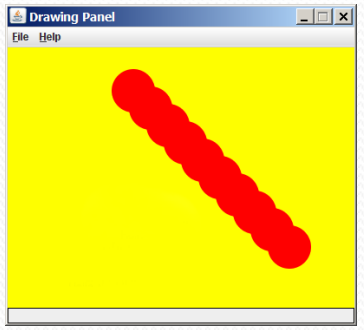
The fun stuff:

- Using loops and parameters to make cool and useful pictures

Drawing with loops

- The *x*,*y*, *w*,*h* expression can use a loop counter variable:

```
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);
}
```



- Nested loops are okay as well:

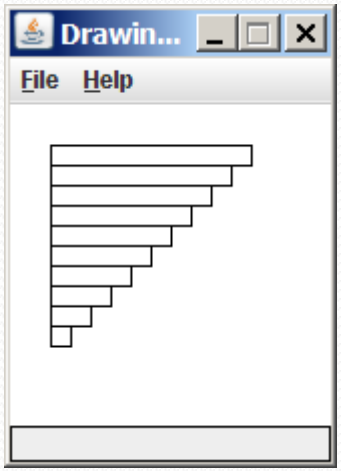
```
DrawingPanel panel = new DrawingPanel(250, 250);
Graphics g = panel.getGraphics();
g.setColor(Color.BLUE);

for (int x = 1; x <= 4; x++) {
    for (int y = 1; y <= 9; y++) {
        g.drawString("Java", x * 40, y * 25);
    }
}
```

Loops that begin at 0

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

- 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);
}
```



Loops mini-exercise

- Modify the stacked rectangles program to draw ten stacked rectangles starting at (20, 20), height 10, each one with width 100. (So the width doesn't change.)

- Width-changing version

```
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);
}
```

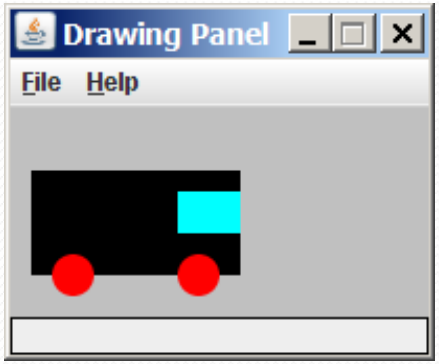
Loops mini-exercise - solution

```
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);
}
```

Superimposing shapes

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

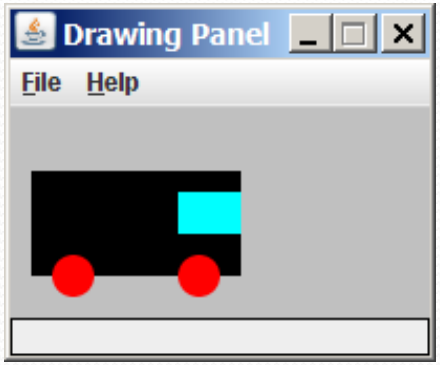
```
import java.awt.*;

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 methods

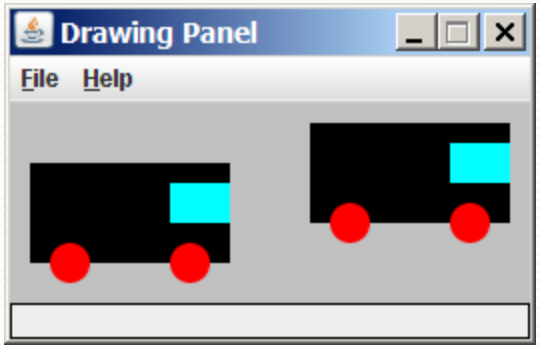
- To draw in multiple methods, you must pass `Graphics g`.

```
import java.awt.*;

public class Car2 {
    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);
    }
}
```



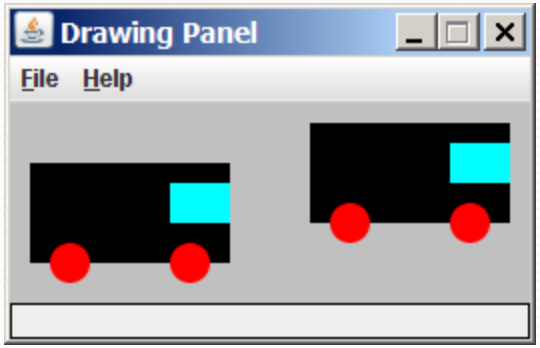
Parameterized figures

- Modify the car-drawing method so that it can draw cars at different positions, as in the following image.
  - Top-left corners: (10, 30), (150, 10)



Parameterized answer

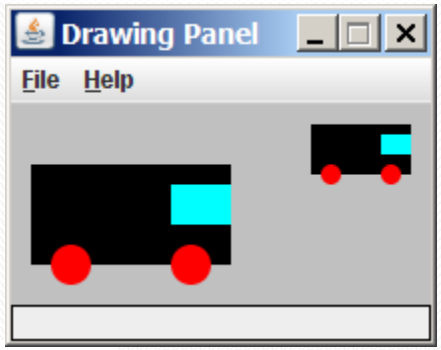
```
import java.awt.*;

public class Car3 {
    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);
    }
}
```

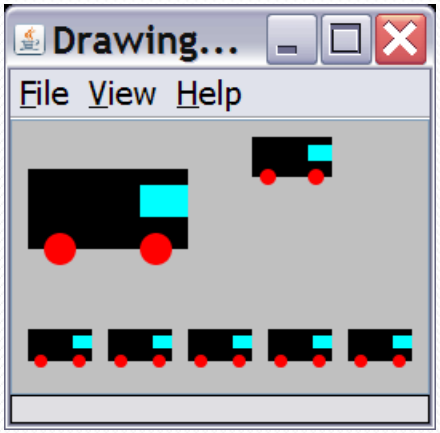


Drawing parameter question

- Modify `drawCar` to allow the car to be drawn at any size.
  - Existing car: size 100
  - Second car: size 50, top/left at (150, 10)

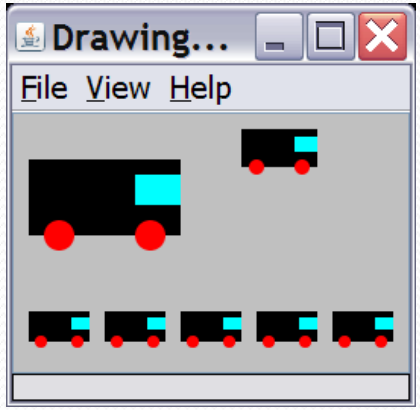
- Then use a `for` loop to draw a line of cars.
  - Start at (10, 130), each car size 40, separated by 50px.





Drawing parameter answer

```
import java.awt.*;

public class Car4 {
    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);
    }
}
```



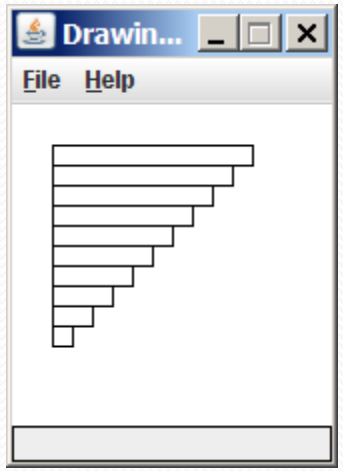
Extra exercises

What follows are a couple exercises related to the slides that you can try on your own (not done in class)

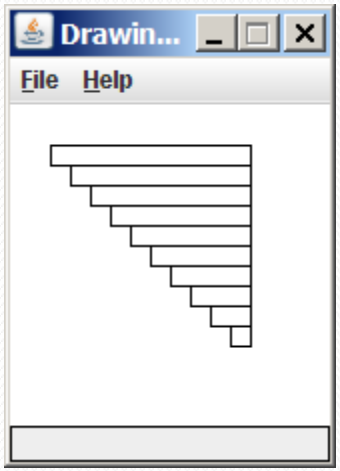
More Drawing w/ loops questions

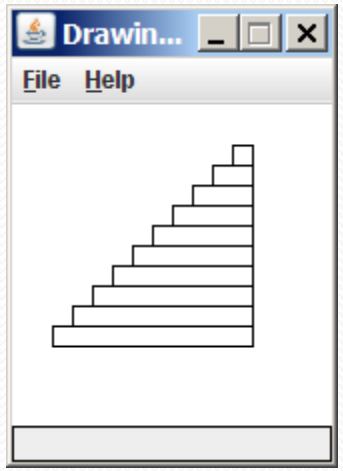
- Code from earlier slide:

```
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 above program that draw the figures at right as output.

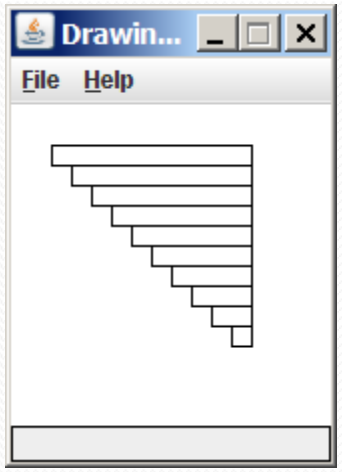




Drawing w/ loops answers

- 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);
}
```