

# CS 106A, Lecture 11

## Graphics

reading:

*Art & Science of Java*, 9.1-9.3

# Plan For Today

- Announcements
- Recap: File Reading
- GraphicsProgram
- Graphical Objects
- Practice: Car

# Announcements

- Read the rest of the slides from yesterday and try the Election practice problem
- Assignment 3 is out—demo coming soon!

# Plan For Today

- Announcements
- **Recap: File Reading**
- GraphicsProgram
- Graphical Objects
- Practice: Car

# File Reading Overview

1. Make a Scanner to open a file to read

```
Scanner input = new Scanner(new File("data.txt"));
```

2. Use Scanner methods such as `nextLine` or `next` to read in the file, usually in a loop while some variation of `hasNext` is true
3. Scanner operations on files are “dangerous” because they dependent on outside resources, so we need to use a try/catch block
4. Close the Scanner when you are done: `input.close()`

# File Reading Overview

1. Make a Scanner to open a file to read

```
Scanner input = new Scanner(new File("data.txt"));
```

2. Use Scanner methods such as `nextLine` or `next` to read in the file, usually in a loop while some variation of `hasNext` is true
3. Scanner operations on files are “dangerous” because they dependent on outside resources, so we need to use a try/catch block
4. Close the Scanner when you are done: `input.close()`

# Scanner methods

Method	Description
<code>sc.nextLine()</code>	reads and returns a one- <i>line</i> String from the file
<code>sc.next()</code>	reads and returns a one-word String from the file
<code>sc.nextInt()</code>	reads and returns an <code>int</code> from the file
<code>sc.nextDouble()</code>	reads and returns a <code>double</code> from the file
<code>sc.hasNextLine()</code>	returns <code>true</code> if there are any more lines
<code>sc.hasNext()</code>	returns <code>true</code> if there are any more tokens
<code>sc.hasNextInt()</code>	returns <code>true</code> if there is a next token and it's an <code>int</code>
<code>sc.hasNextDouble()</code>	returns <code>true</code> if there is a next token and it's a <code>double</code>
<code>sc.close();</code>	should be called when done reading the file

# File Reading Overview

1. Make a Scanner to open a file to read

```
Scanner input = new Scanner(new File("data.txt"));
```

2. Use Scanner methods such as `nextLine` or `next` to read in the file, usually in a loop while some variation of `hasNext` is true
3. Scanner operations on files are “dangerous” because they dependent on outside resources, so we need to use a try/catch block
4. Close the Scanner when you are done: `input.close()`



# Try/Catch

```
try {  
    statements;    // code that might throw an exception  
} catch (ExceptionType name) {  
    statements;    // code to handle the error  
}
```

- To execute code that might throw an exception, you must enclose it in a try/catch statement.

```
try {  
    Scanner input = new Scanner(new File("data.txt"));  
    ...  
} catch (IOException ex) {  
    println("Error reading the file: " + ex);  
}
```

# Try/Catch

To execute code that might throw an exception, you must enclose it in a try/catch statement.



If something  
fails up here...

```
try {  
    Scanner input = new Scanner(new File("data.txt"));  
    while (input.hasNextLine()) {  
        String line = input.nextLine();  
        println(line);  
    }  
} catch (FileNotFoundException ex) {  
    println("Error reading the file: " + ex);  
}
```

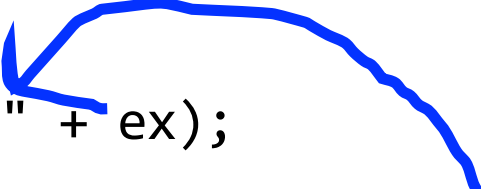
# Try/Catch

To execute code that might throw an exception, you must enclose it in a try/catch statement.

If something  
fails up here...



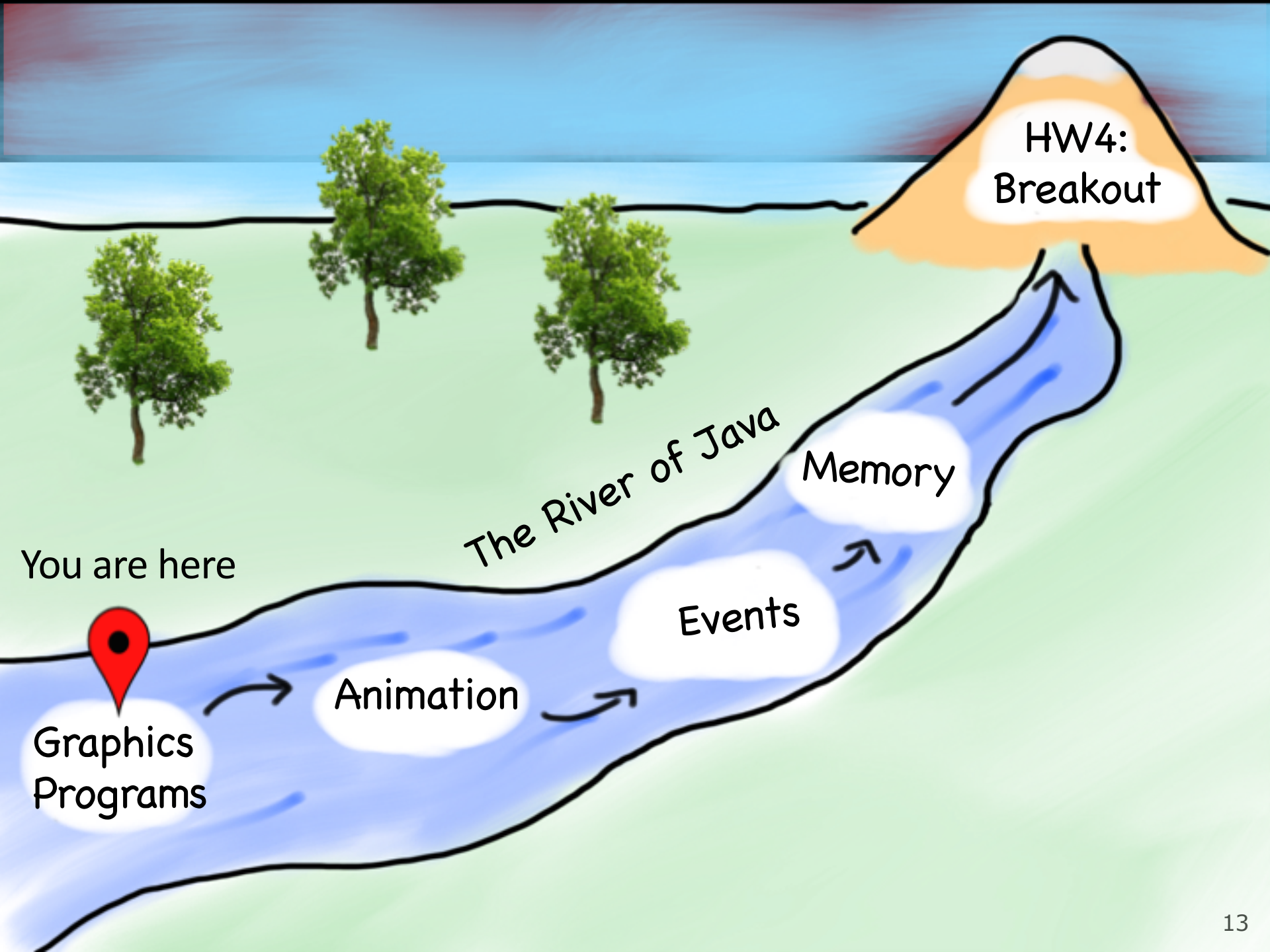
```
try {  
    Scanner input = new Scanner(new File("data.txt"));  
    while (input.hasNextLine()) {  
        String line = input.nextLine();  
        println(line);  
    }  
} catch (FileNotFoundException ex) {  
    println("Error reading the file: " + ex);  
}
```



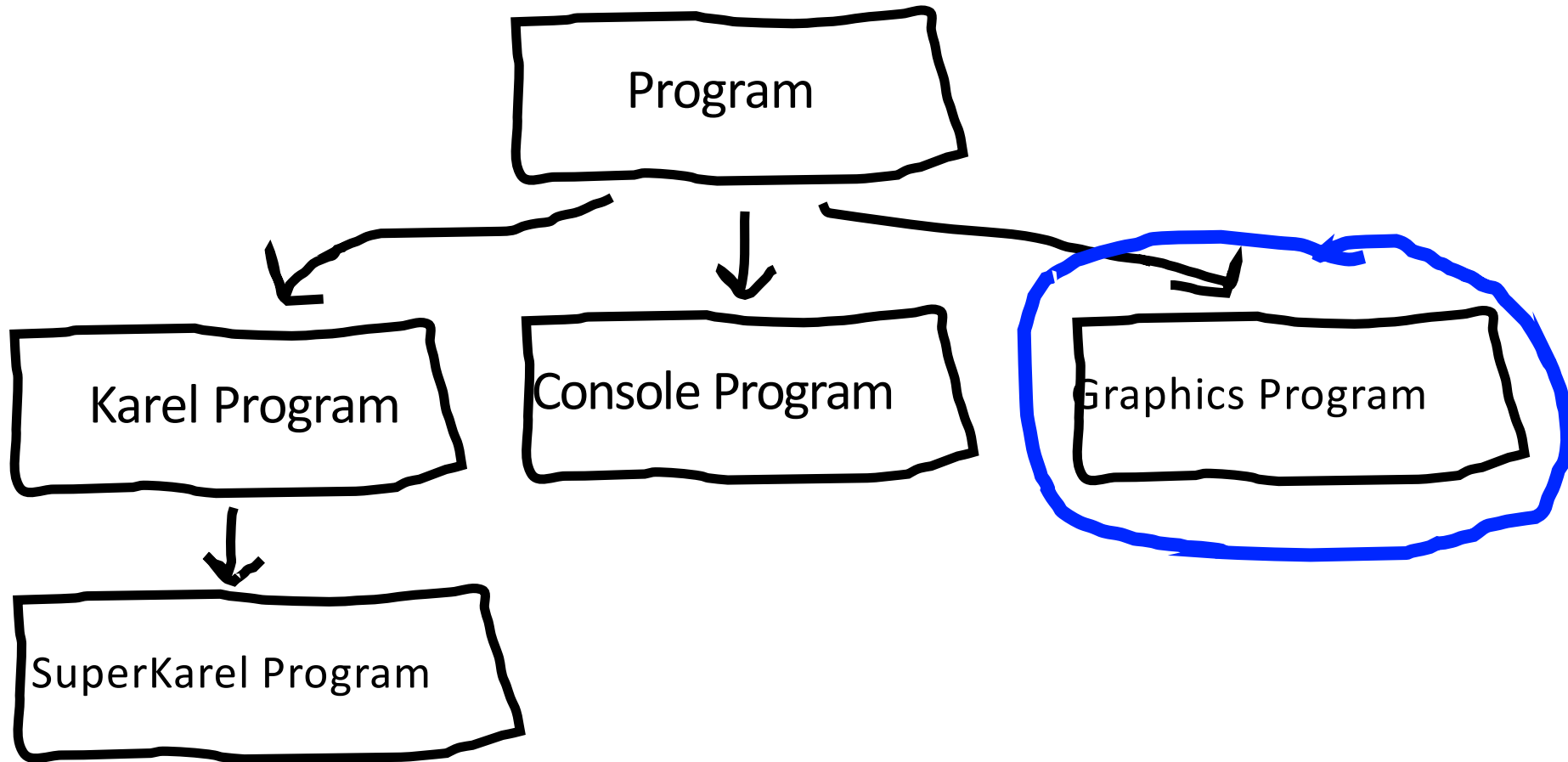
... we immediately jump  
down here.

# Plan For Today

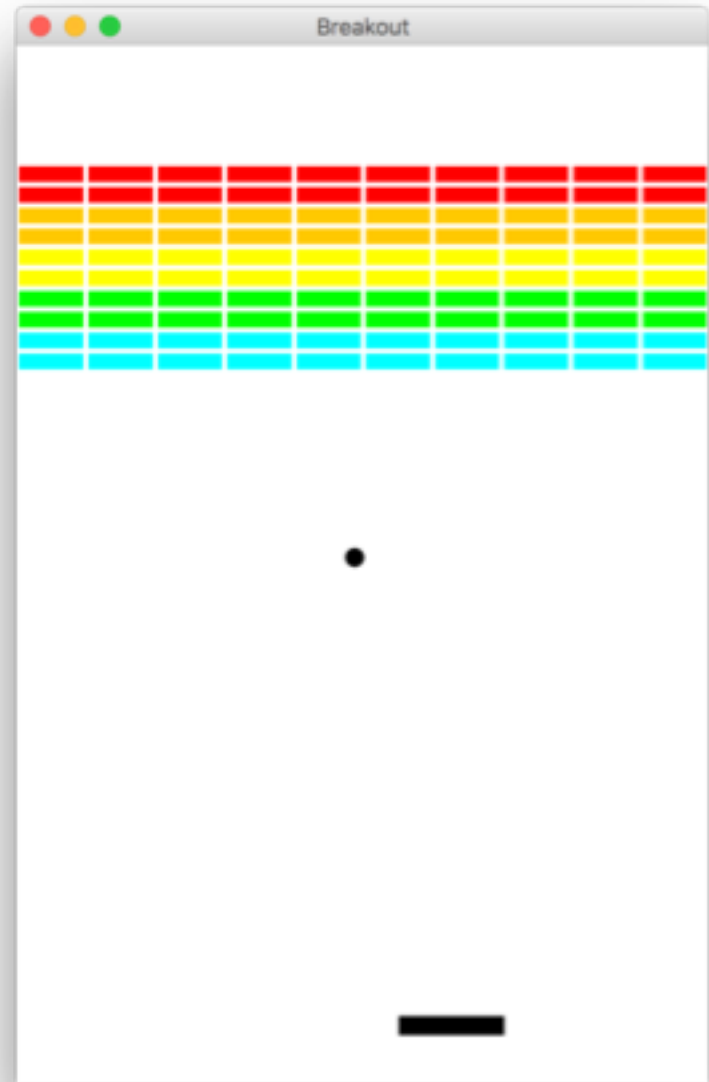
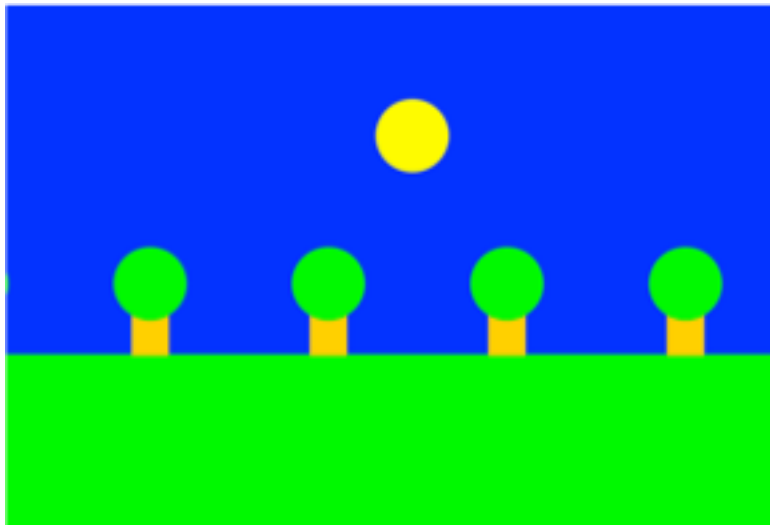
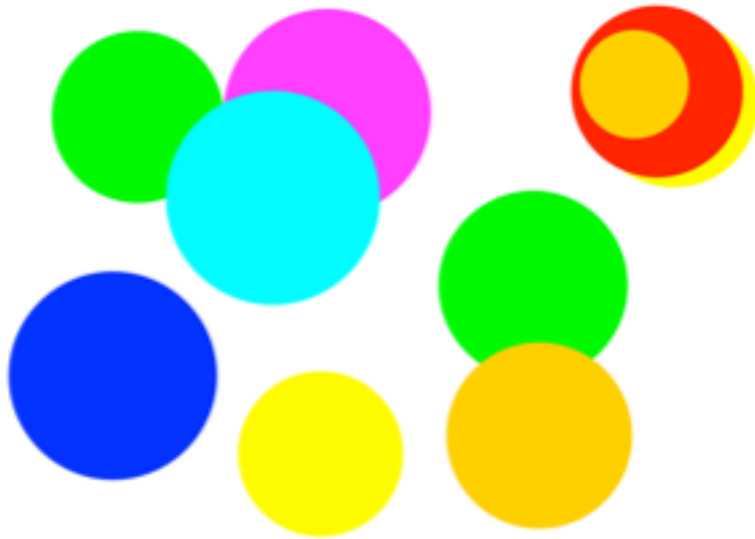
- Announcements
- Recap: File Reading
- **GraphicsProgram**
- Graphical Objects
- Practice: Car



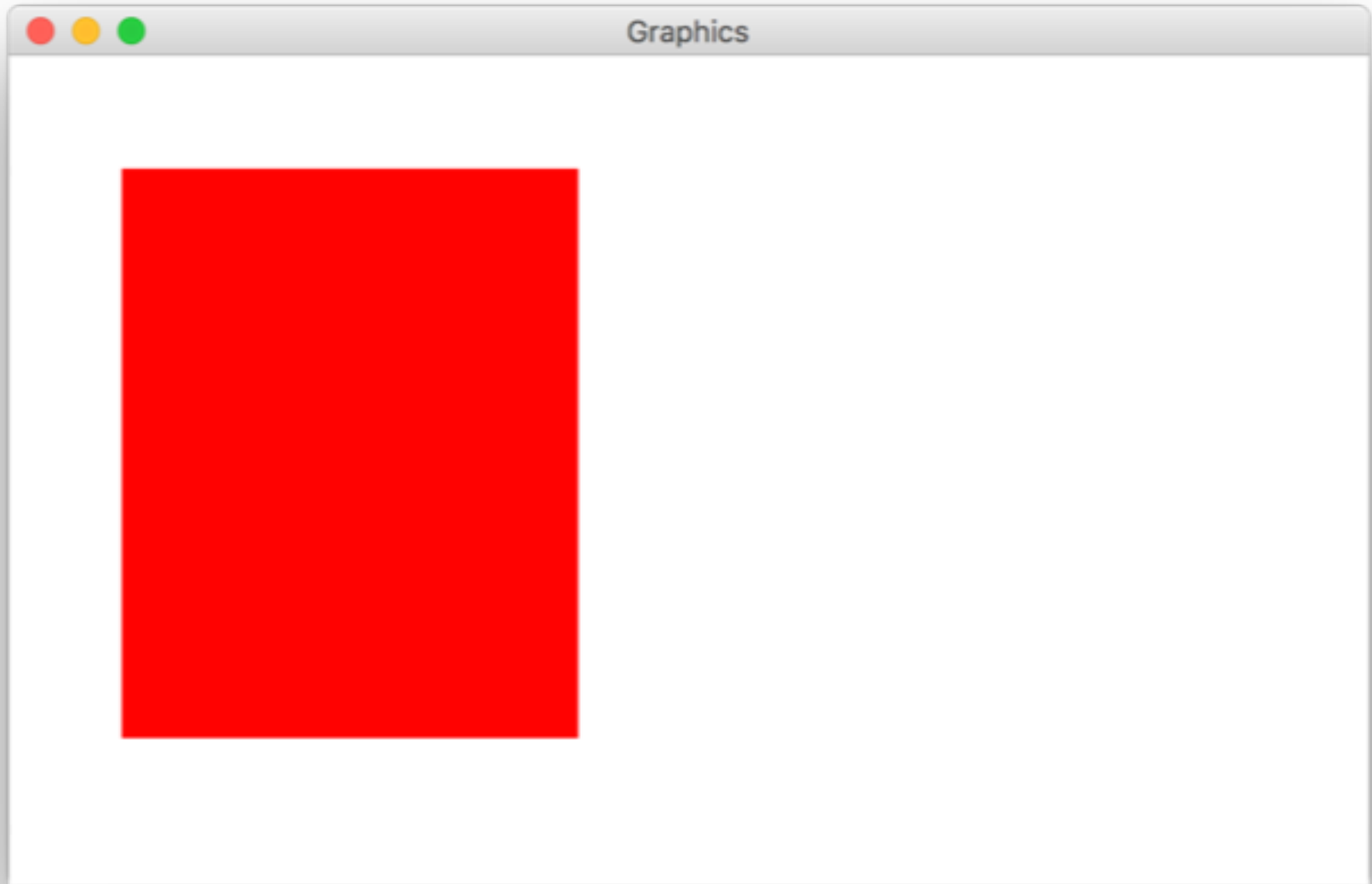
# Java



# Graphics Programs



# Our First GraphicsProgram





# Our First GraphicsProgram

```
import acm.program.*;
import acm.graphics.*; // Stanford graphical objects
import java.awt.*;      // Java graphical objects

public class MyGraphics extends GraphicsProgram {
    public void run() {
        GRect rect = new GRect(50, 50, 200, 250);
        rect.setFilled(true);
        rect.setColor(Color.RED);
        add(rect);
    }
}
```

# Our First GraphicsProgram

```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

// Set some properties
rect.setFilled(true);
rect.setColor(Color.RED);

// Add to the canvas
add(rect);
```

# Our First GraphicsProgram

```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

// Set some properties
rect.setFilled(true);
rect.setColor(Color.RED);

// Add to the canvas
add(rect);
```

# Our First GraphicsProgram

```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

// Set some properties
rect.setFilled(true);
rect.setColor(Color.RED);

// Add to the canvas
add(rect);
```

# Our First GraphicsProgram

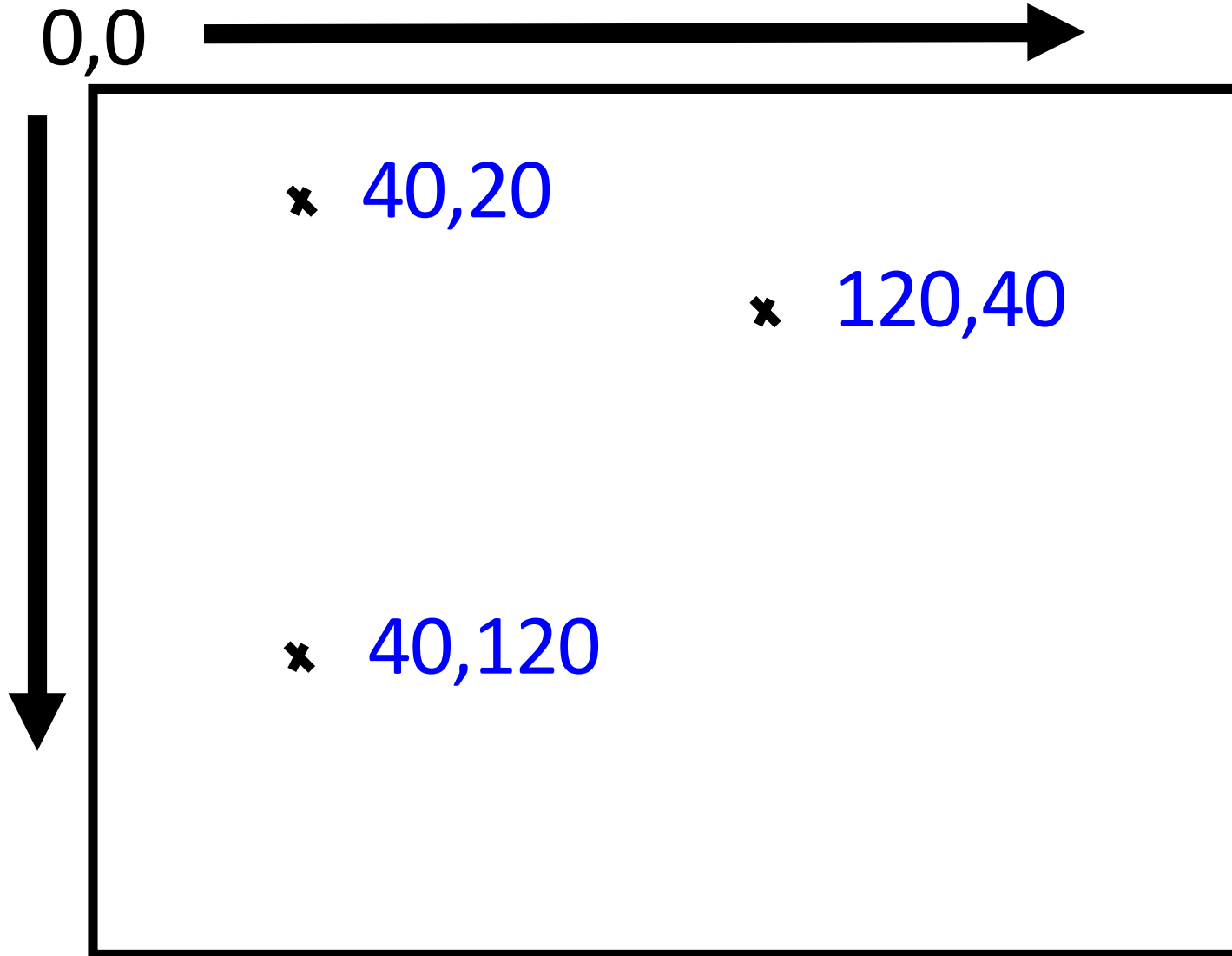
```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

// Set some properties
rect.setFilled(true);
rect.setColor(Color.RED);

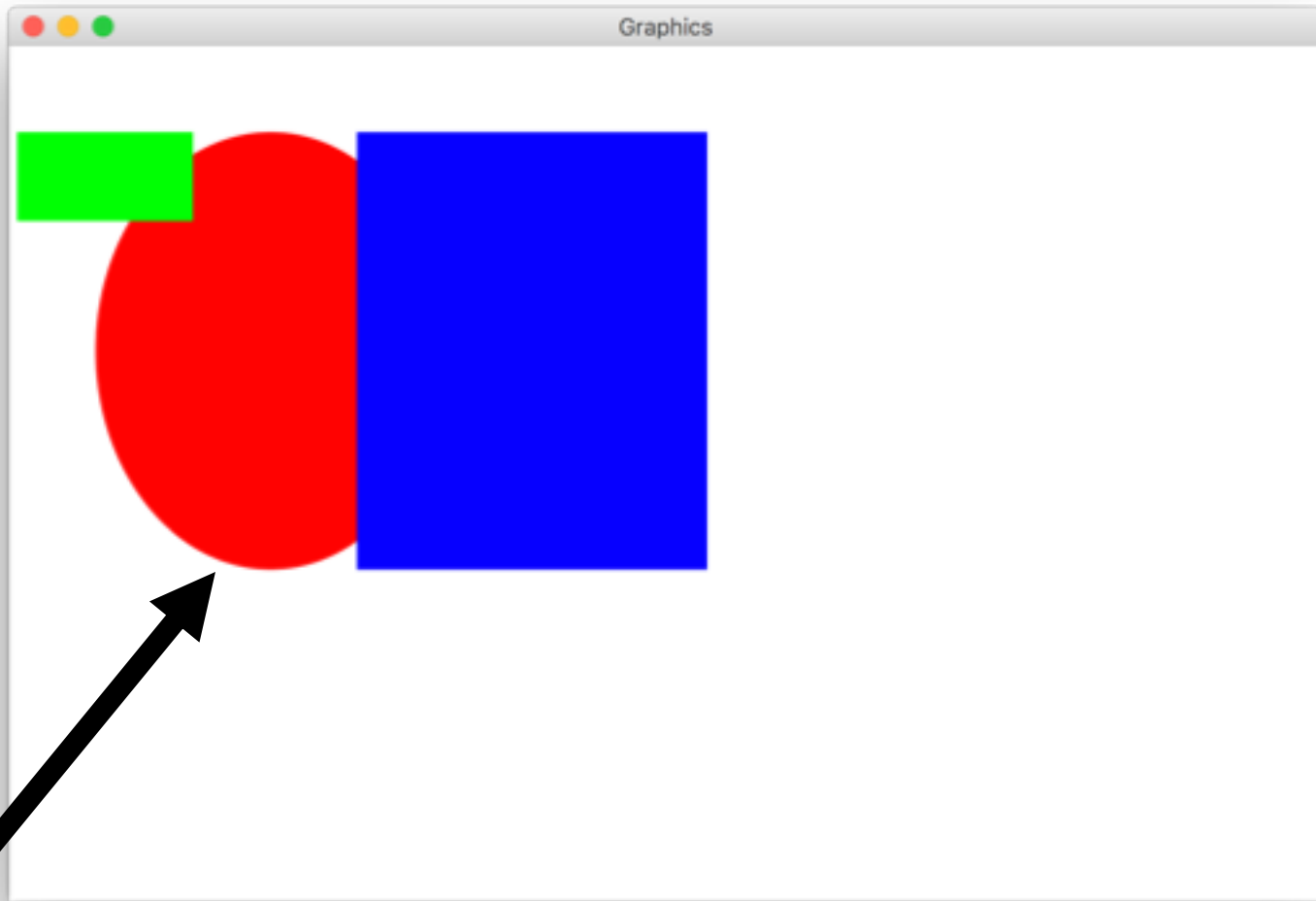
// Add to the canvas
add(rect);
```



# The Graphics Canvas



# Collage Model



Must have been added first

# Plan For Today

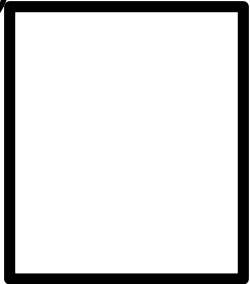
- Announcements
- Recap: File Reading
- GraphicsProgram
- **Graphical Objects**
- Practice: Cars and Checkerboards



# Graphical Objects

GRect

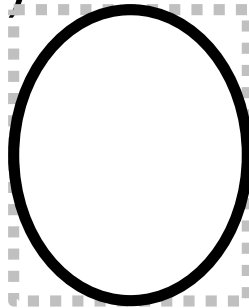
(x, y)



(x+w,  
y+h)

GOval

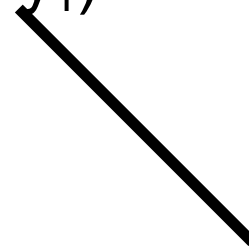
(x, y)



(x+w,  
y+h)

GLine

(x<sub>1</sub>, y<sub>1</sub>)



(x<sub>2</sub>, y<sub>2</sub>)

GLabel

*Hello there!*

GImage



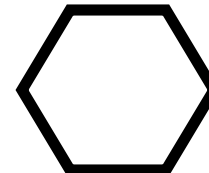
GArc



GRoundRect



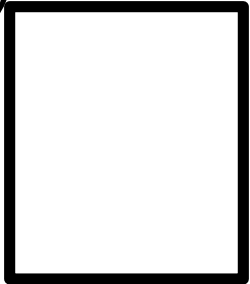
GPolygon



# Graphical Objects

**GRect**

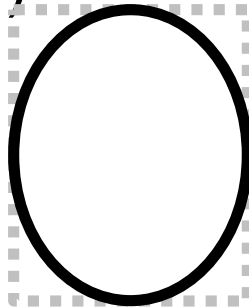
$(x, y)$



$(x+w, y+h)$

**GOval**

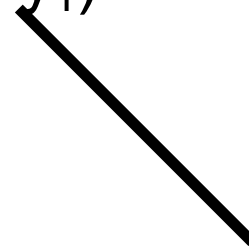
$(x, y)$



$(x+w, y+h)$

**GLine**

$(x_1, y_1)$



$(x_2, y_2)$

**GLabel**

*Hello there!*

**GImage**



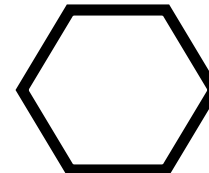
**GArc**



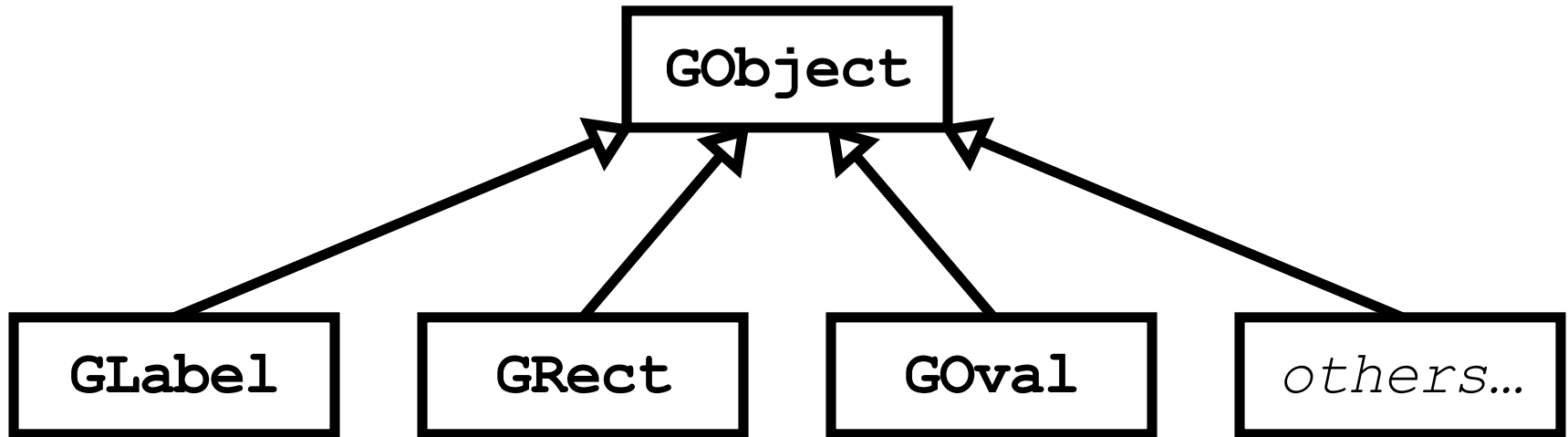
**GRoundRect**



**GPolygon**



# Graphical Objects



```
GRect myRect = new GRect(50, 50, 350, 270);
```

# Primitives vs. Objects

## Primitive Variable Types

**int**  
**double**  
**char**  
**boolean**

## Object Variable Types

**GRect**  
**G Oval**  
**GLine**  
**Scanner**  
...

Object variables:

1. Have UpperCamelCase types
2. You can call methods on them
  - Uses “dot syntax”
3. Are constructed using **new**

# Methods on Graphics Objects

We manipulate graphics objects by calling methods on them:

**object.method(parameters);**

Receiver Message

# Methods on Graphics Objects

We manipulate graphics objects by calling methods on them:

**object** . **method** ( **parameters** ) ;

└────────┘    └────────┘    └────────────────┘

Who?                      What?                      What specifically?

Example:

**rect** . **setColor** ( **Color** . **RED** ) ;

# GObject Methods

The following operations apply to all **GObjects**:

*object*.**setColor** (*color*)

Sets the color of the object to the specified color constant.

*object*.**setLocation** (*x*, *y*)

Changes the location of the object to the point (*x*, *y*).

*object*.**move** (*dx*, *dy*)

Moves the object on the screen by adding *dx* and *dy* to its current coordinates.

*object*.**getWidth** ()

Returns the width of the object

*object*.**getHeight** ()

Returns the height of the object

# Colors

- Specified as predefined Color constants:

Color.*NAME* , where *NAME* is one of:



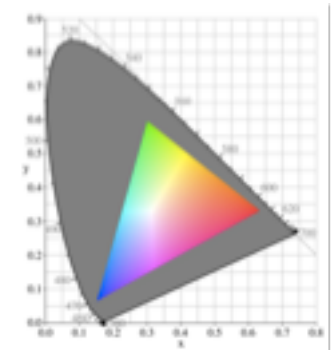
BLACK	BLUE	CYAN	DARK_GRAY	GRAY
GREEN	LIGHT_GRAY	MAGENTA	ORANGE	PINK
RED	WHITE	YELLOW		

```
rect.setColor(Color.MAGENTA);
```

- Or create one using Red-Green-Blue (RGB) values of 0-255  
new Color(*red*, *green*, *blue*)

– Example:

```
rect.setColor(new Color(192, 128, 64));
```





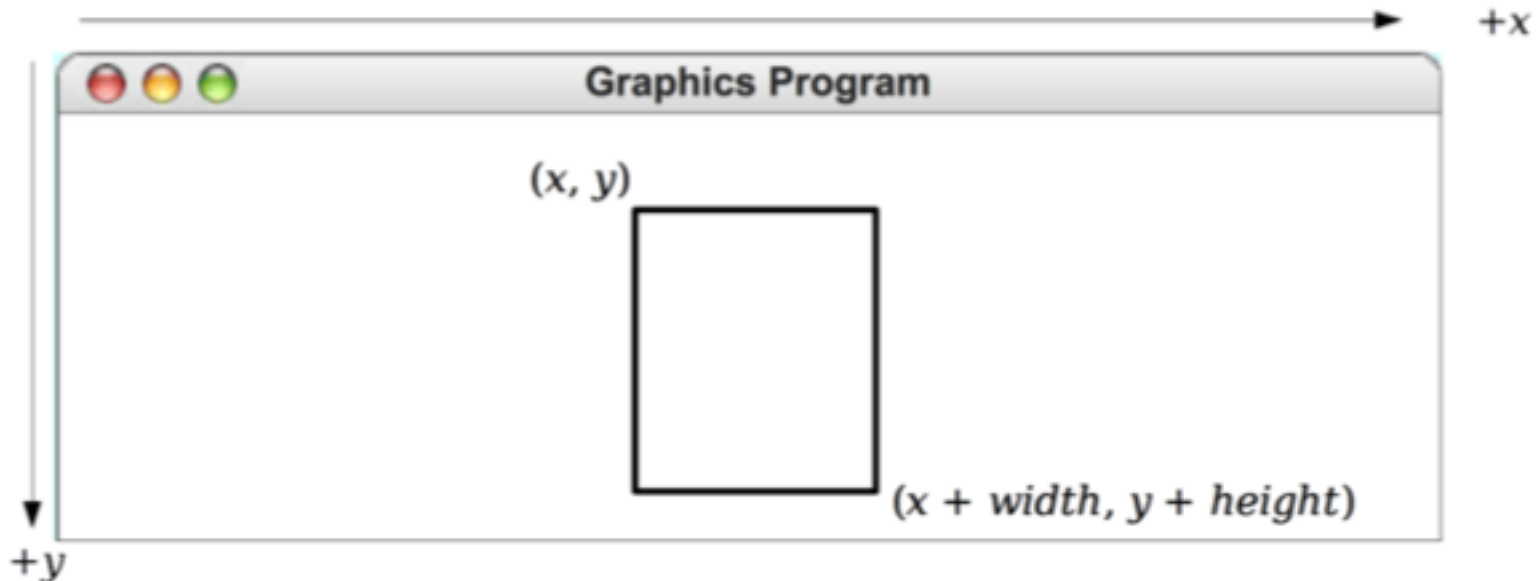
# GRect

**`new GRect(x, y, width, height);`**

- Creates a rectangle with the given width and height, whose upper-left corner is at (*x*, *y*)

**`new GRect(width, height);`**

- Same as above, but defaults to (*x*, *y*) = (0, 0)



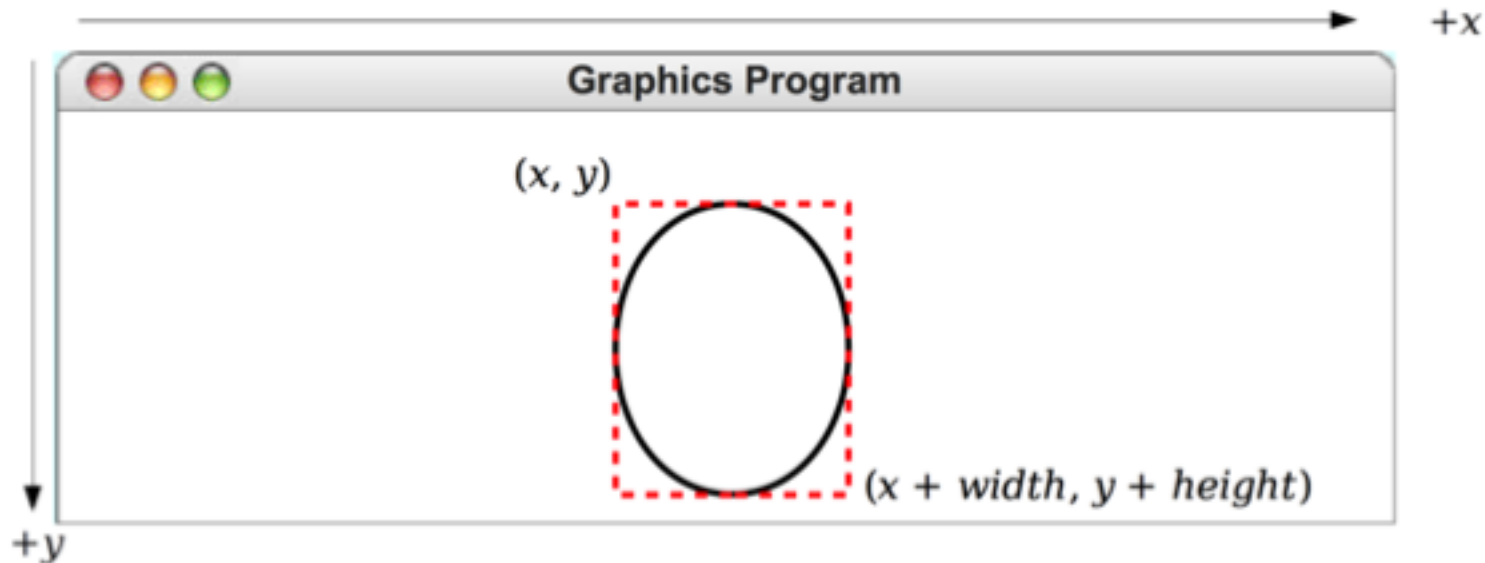
# G Oval

**`new GOval(x, y, width, height);`**

- Creates an oval that fits inside a rectangle with the given width and height, and whose upper-left corner is at (x, y)

**`new GOval(width, height);`**

- Same as above, but defaults to (x, y) = (0, 0)



# GRect and GOval

Methods shared by the **GRect** and **GOval** classes

*object*.**setFilled**(*fill*)

If *fill* is **true**, fills in the interior of the object; if **false**, shows only the outline.

*object*.**setFillColor**(*color*)

Sets the color used to fill the interior, which can be different from the border.

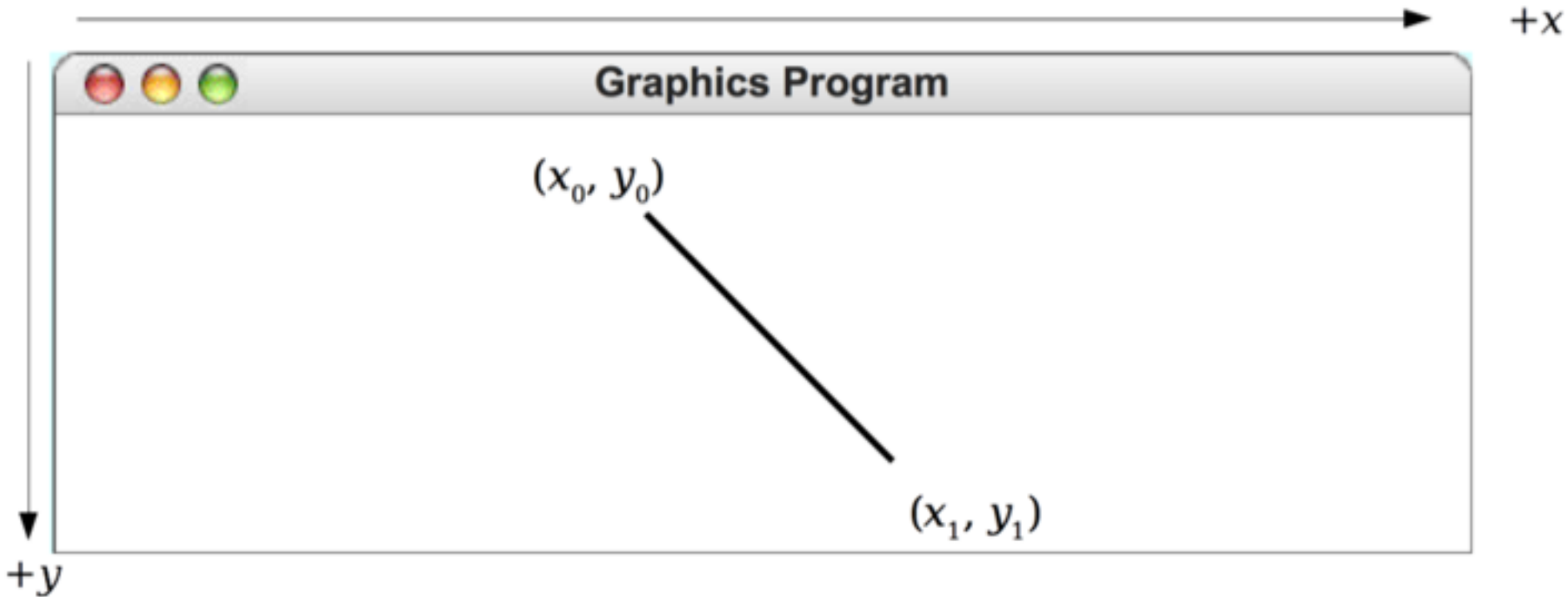
*object*.**setSize**(*width*, *height*)

Sets the object's size to be the given width and height

# GLine

```
new GLine( $x_0$ ,  $y_0$ ,  $x_1$ ,  $y_1$ );
```

- Creates a line extending from  $(x_0, y_0)$  to  $(x_1, y_1)$



# GLabel

```
new GLabel(“your text here”, x, y);
```

- Creates a label with the given text, whose **baseline** starts at (x, y). NOT positioned according to the top-left corner!

```
new GLabel(“your text here”);
```

- Same as above, but defaults to (x, y) = (0, 0)



# GLabel Methods

Methods specific to the **GLabel** class

*label*.**getDescent**()

Returns the height of the label below its baseline.

*label*.**getAscent**()

Returns the height of the label above its baseline.

*label*.**setFont**(*font*)

Sets the font used to display the label as specified by the font string.

The font is typically specified as a string in the form

*"family-style-size"*

*family* is the name of a font family

*style* is either **PLAIN**, **BOLD**, **ITALIC**, or **BOLDITALIC**

*size* is an integer indicating the point size

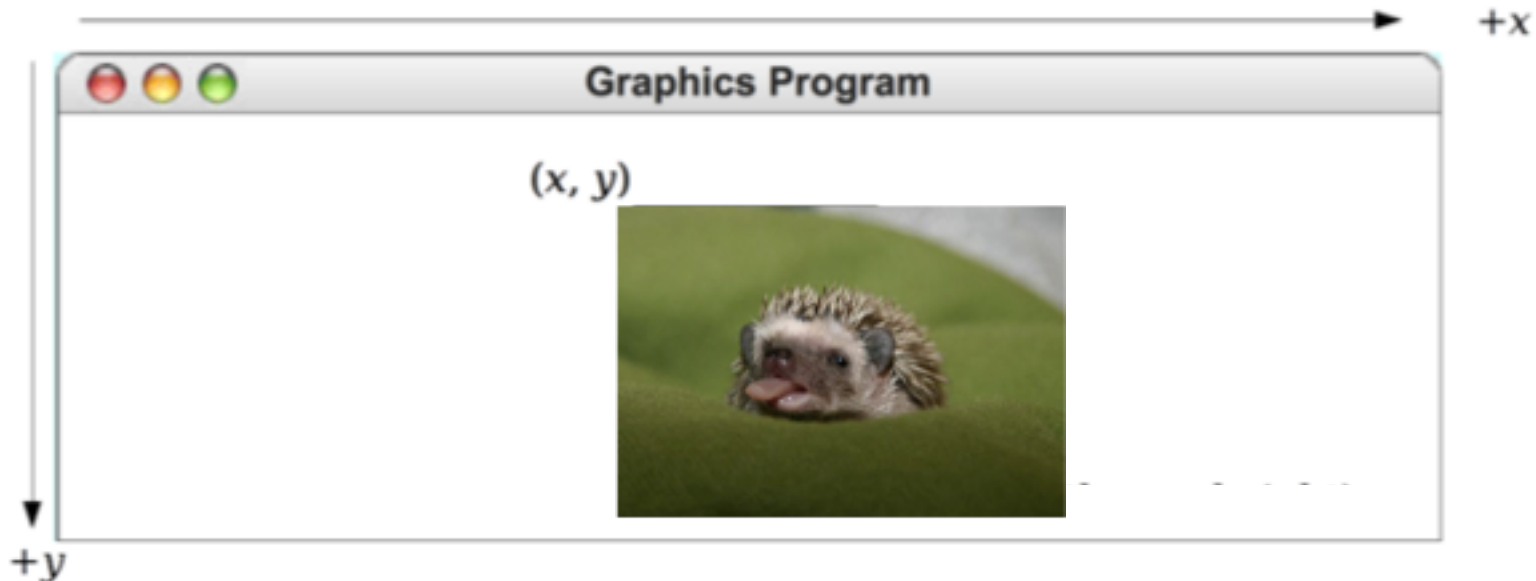
# GImage

**`new GImage("your filename here", x, y);`**

- Creates a an image displaying the given file, whose upper-left corner is at (x, y)

**`new GImage("your filename here");`**

- Same as above, but defaults to (x, y) = (0, 0)



# GImage Methods

*object*.**setSize**( *width*, *height* )

Sets the object's size to be the given width and height



# GraphicsProgram Methods

- GraphicsProgram contains these useful methods:

Method	Description
add( <i>gobj</i> ); add( <i>gobj</i> , <i>x</i> , <i>y</i> );	adds a graphical object to the window
getElementAt( <i>x</i> , <i>y</i> )	return the object at the given (x,y) position(s)
getElementCount()	return number of graphical objects onscreen
getWidth(), getHeight()	return dimensions of window
remove( <i>gobj</i> );	removes a graphical object from the window
removeAll();	remove all graphical objects from window
setCanvasSize( <i>w</i> , <i>h</i> );	set size of drawing area
setBackground( <i>color</i> );	set window's background color

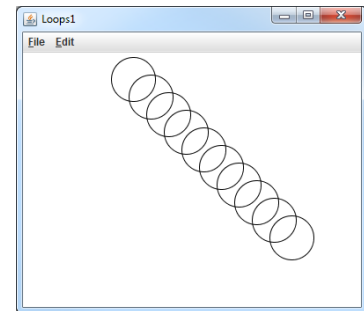
# Plan For Today

- Announcements
- Recap: File Reading
- GraphicsProgram
- Graphical Objects
- **Practice: Car**

# Practice: Drawing with Loops

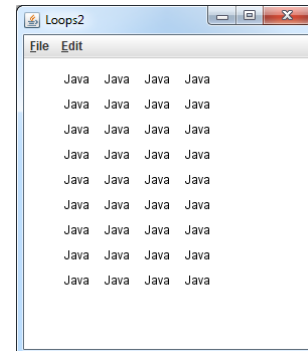
- The *x, y, width, height* expressions can use the loop counter variable:

```
for (int i = 0; i < 10; i++) {  
    add(new G Oval(100 + 20 * i, 5 + 20 * i, 50, 50));  
} //           x           y           w           h
```



- Nested loops can be used with graphics:

```
for (int x = 1; x <= 4; x++) {  
    for (int y = 1; y <= 9; y++) {  
        add(new GLabel("Java", x * 40, y * 25));  
    }  
}
```

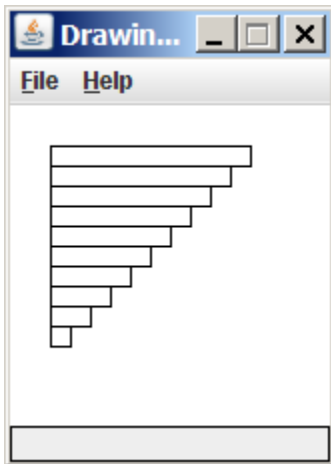


# Practice: Drawing with Loops

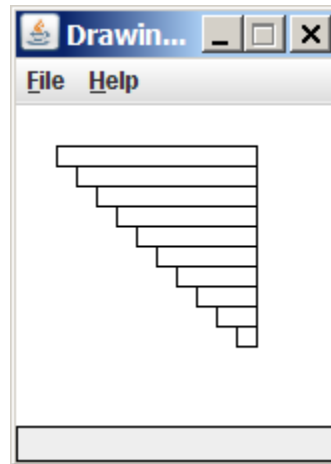
- **Q:** What is the output of the following code?

```
for (int i = 0; i < 10; i++) {  
    add(new GRect(20 + 10 * i, 20 + 10 * i,  
                  100 - 10 * i, 10));  
}
```

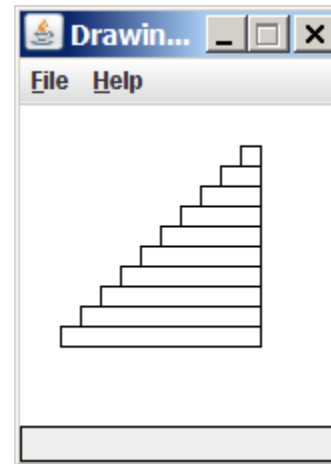
1.



2.



3.



4.

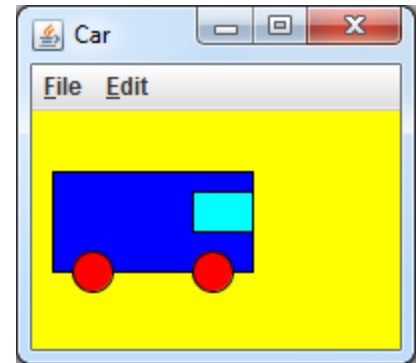
none

– (How would we modify the code above to produce each output?)

# Practice: Car

Write a graphical program named **Car** that draws a figure that looks (kind of) like a car.

- Red wheels at (20, 70) and (80, 70), size 20x20
- Cyan windshield at (80, 40), size 30x20
- Blue body at (10, 30), size 100x50
- Yellow background



# Recap

- Announcements
- Recap: File Reading
- GraphicsProgram
- Graphical Objects
- Practice: Car

**Next time: More Graphics + Animation**