# 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: Cars and Checkerboards

# Plan For Today

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

# 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
3. Scanner operations on files are ”dangerous”, 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
3. Scanner operations on files are ”dangerous”, so we need to use a try/catch block
4. Close the Scanner when you are done – `input.close()`

# Scanner methods

| Method | Description |
|---|---|
| `sc.nextLine()` | reads and returns a one-*line* `String` from the file |
| `sc.next()` | reads and returns a one-word `String` from the file |
| `sc.nextInt()` | reads and returns an `int` from the file |
| `sc.nextDouble()` | reads and returns a `double` from the file |
| `sc.hasNextLine()` | returns `true` if there are any more lines |
| `sc.hasNext()` | returns `true` if there are any more tokens |
| `sc.hasNextInt()` | returns `true` if there is a next token and it's an `int` |
| `sc.hasNextDouble()` | returns `true` if there is a next token and it's a `double` |
| `sc.close();` | 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
3. Scanner operations on files are ”dangerous”, so we need to use a try/catch block
4. Close the Scanner when you are done – `input.close()`

# Try/Catch

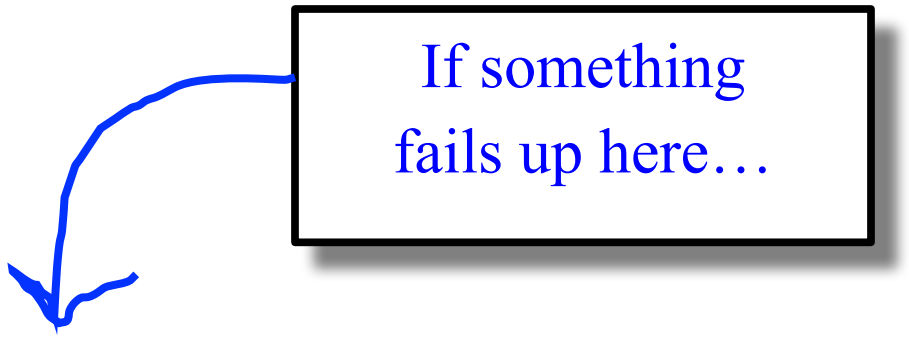
```
try {
    statements;    // code that might throw an exception
} catch (ExceptionType name) {
    statements;    // code to handle the error
}
```

- 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

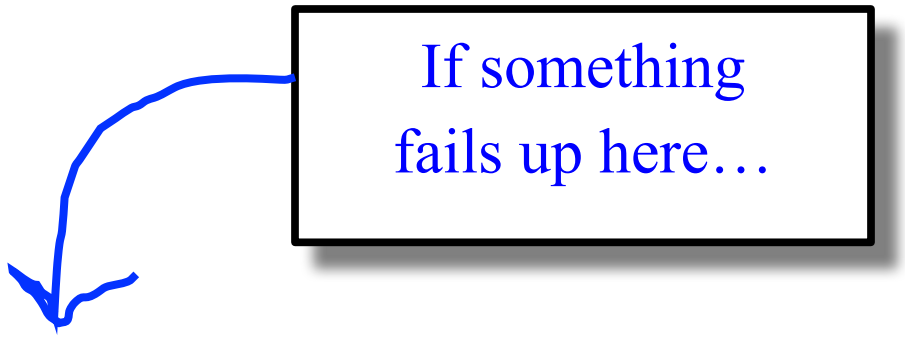
To execute code that might throw an exception, you must enclose it in a `try/catch` statement.



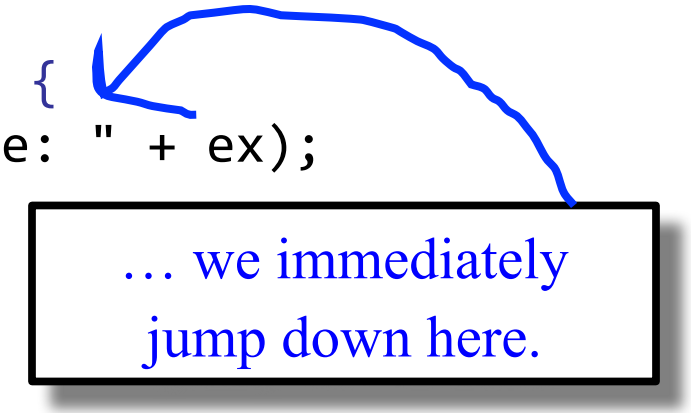

```
try {
    Scanner input = new Scanner(new File("data.txt"));
    while (input.hasNextLine()) {
           String line = input.nextLine();
           println(line);
    }
} catch (FileNotFoundException ex) {
    println("Error reading the file: " + ex);
}
```

# Try/Catch

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




```
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);
}
```



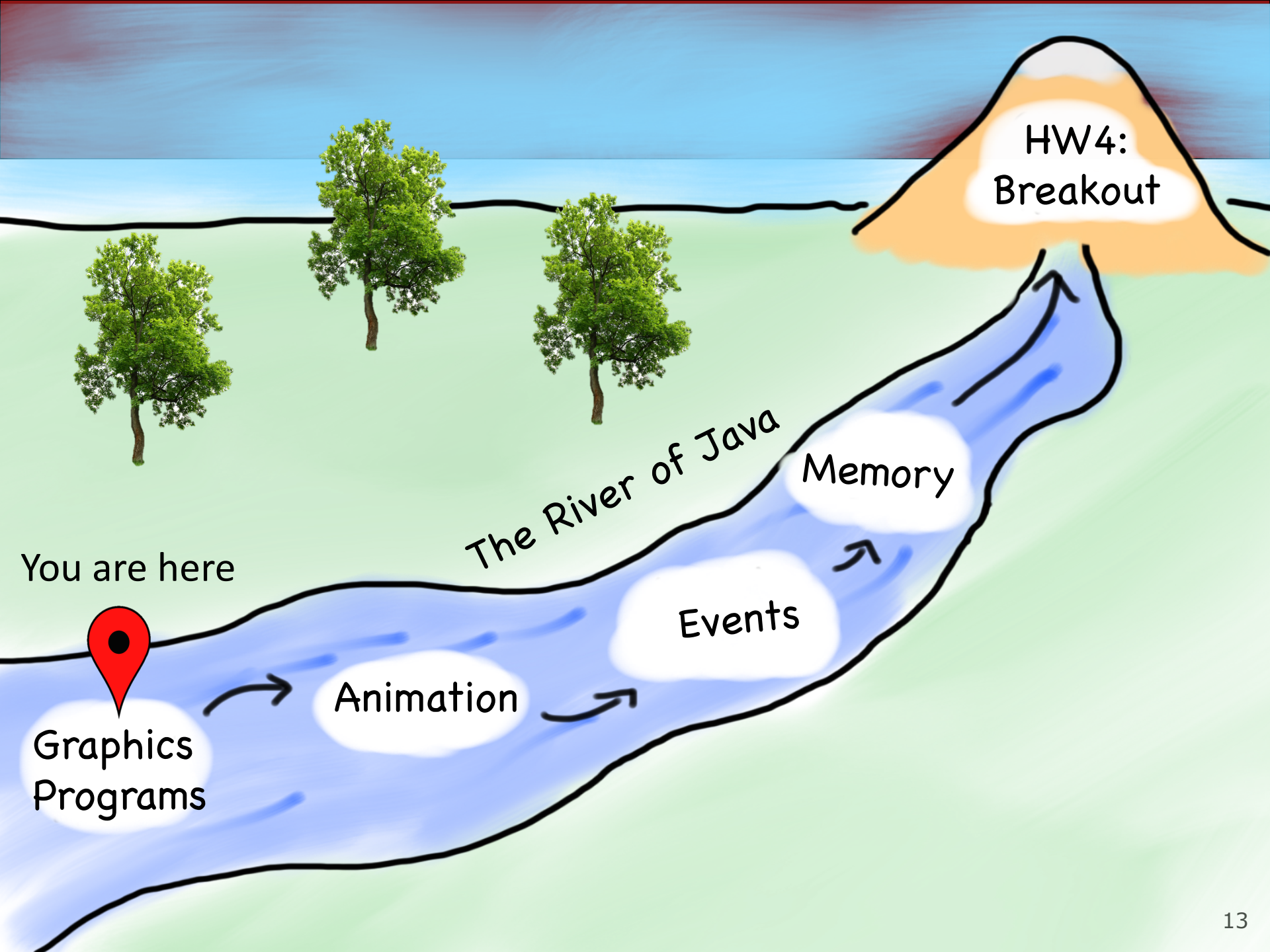
# Mixing lines and tokens

| Input file `input.txt` : | Output to console: |
|---|---|
| `The quick brown fox jumps over`<br>`the lazy dog.` | `Line has 6 words`<br>`Line has 3 words` |

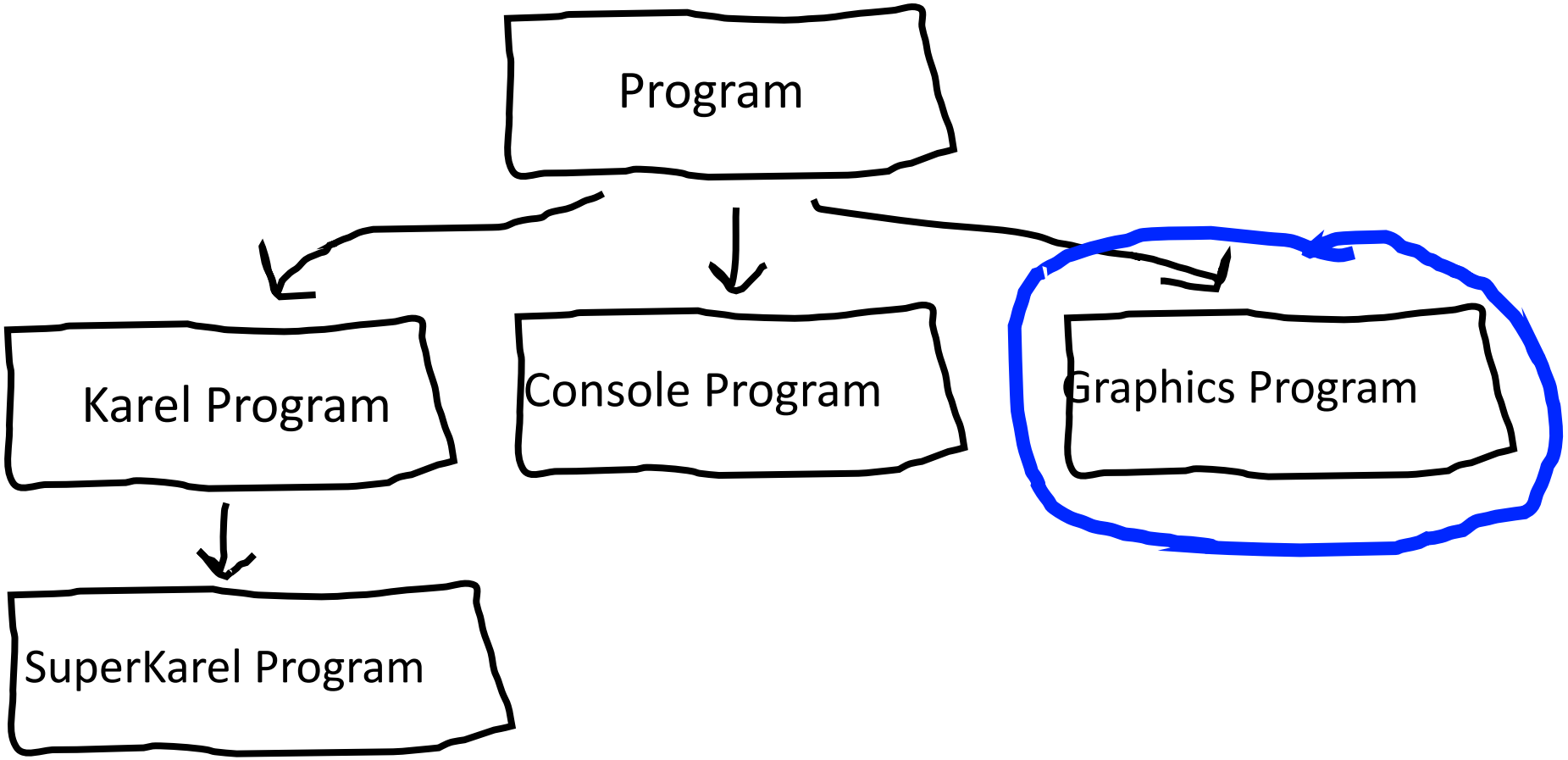
```
// Counts the words on each line of a file
Scanner input = new Scanner(new File("input.txt"));
while (input.hasNextLine()) {
    Scanner tokens = new Scanner(input.nextLine());

    // process the contents of this line
    int count = 0;
    while (tokens.hasNext()) {
        String word = tokens.next();
        count++;
    }
    println("Line has " + count + " words");
}
...
```

# Plan For Today

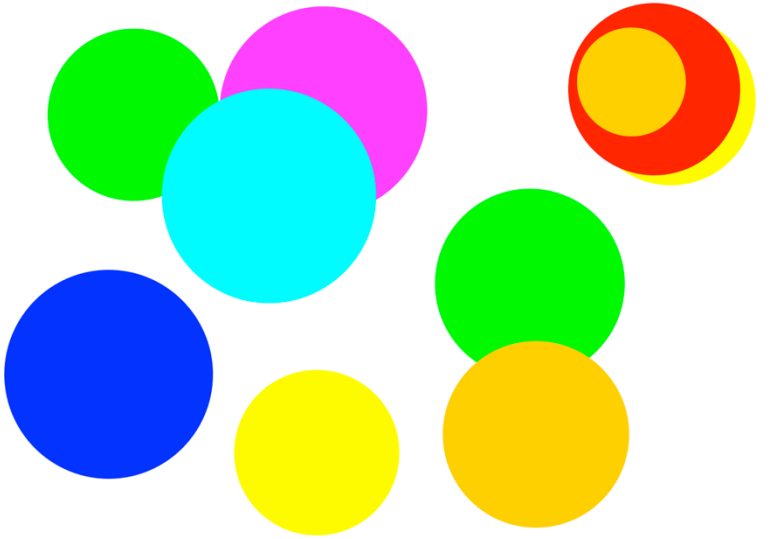
- Announcements
- Recap: File Reading
- GraphicsProgram
- Graphical Objects
- Practice: Cars and Checkerboards

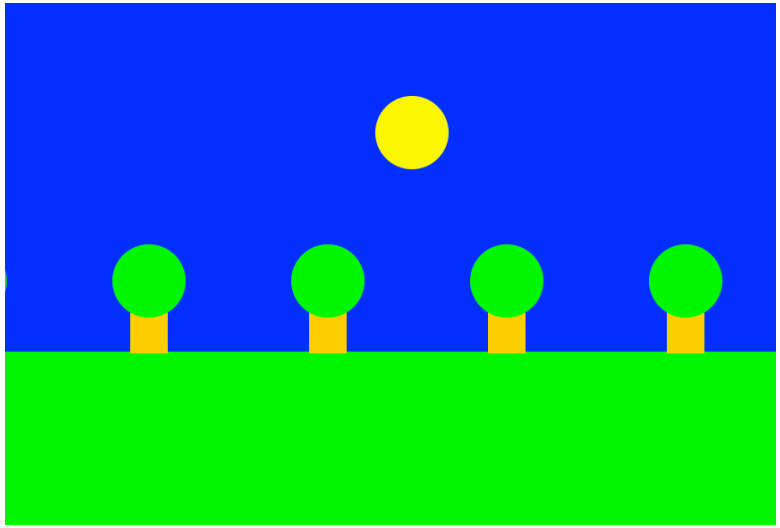

HW4:
Breakout
The River of Java
Memory
You are here
Events
Animation
Graphics
Programs

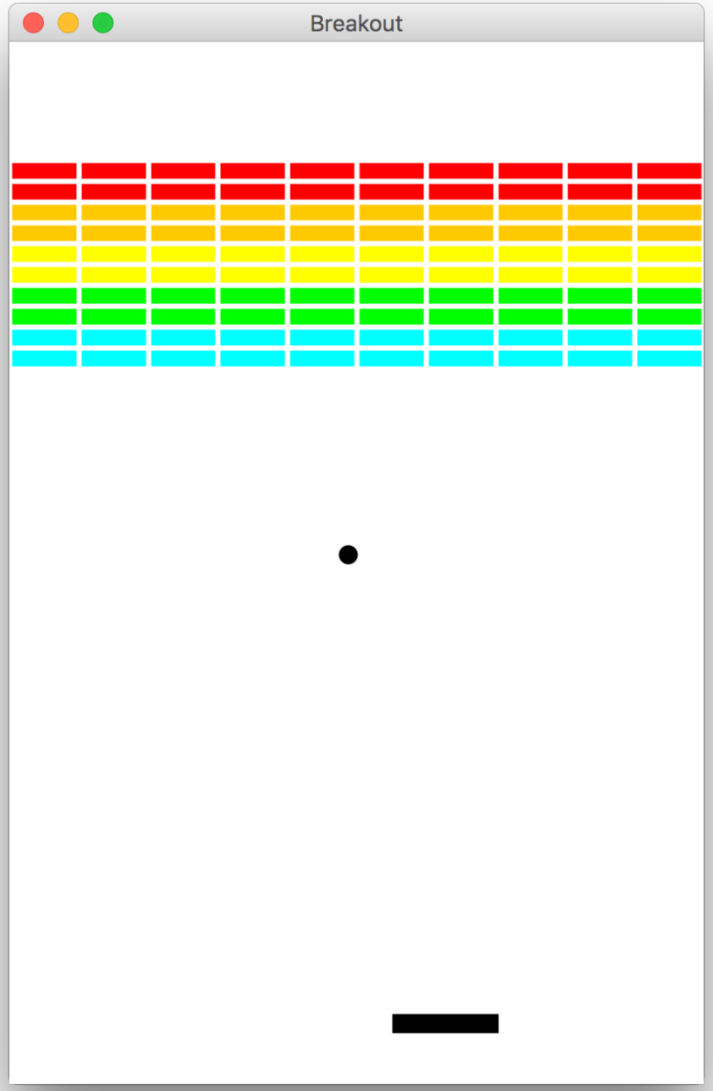
# Java


Program
Karel Program
Console Program
Graphics Program
SuperKarel Program

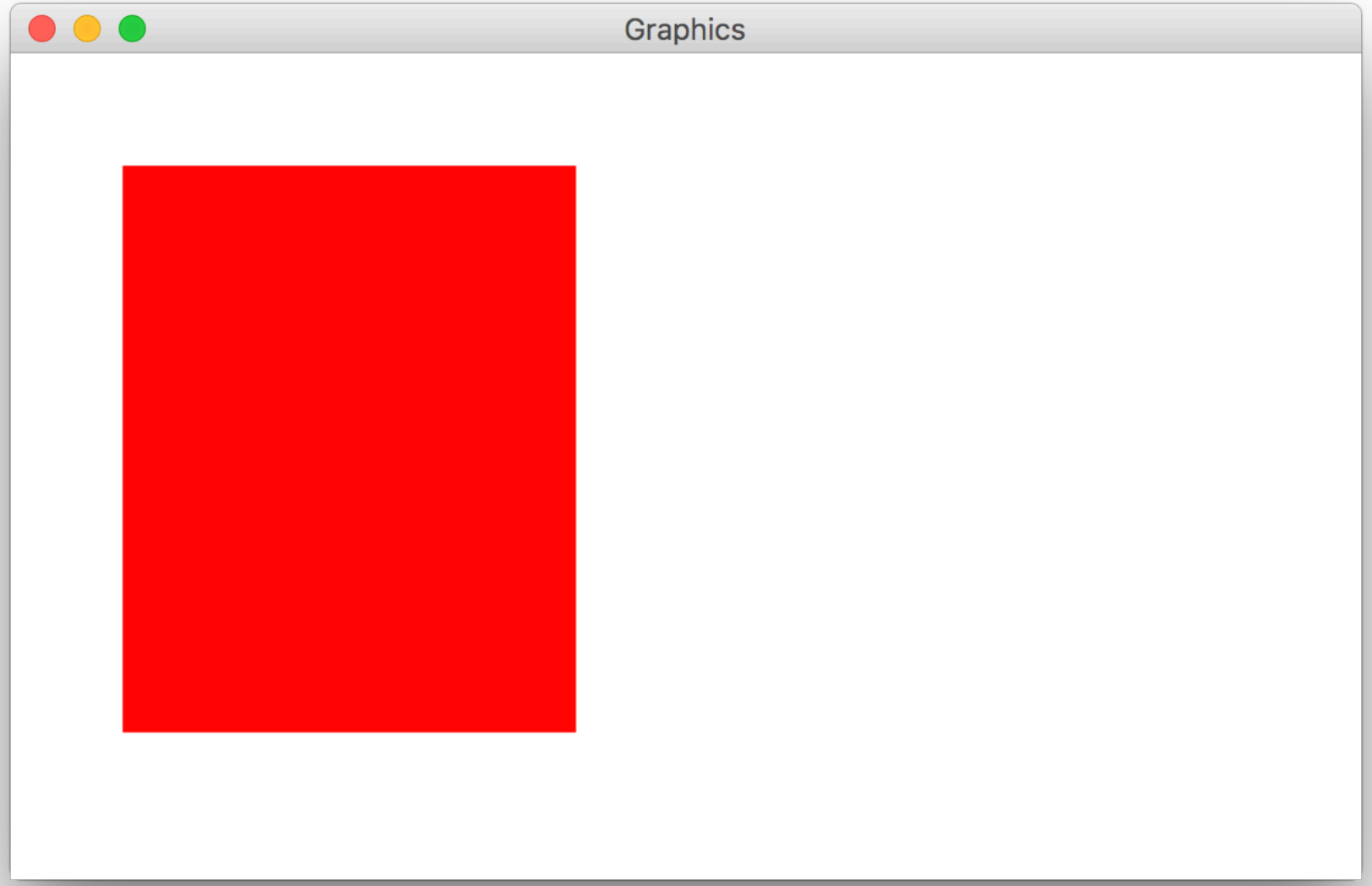
# Graphics Programs







# Our First GraphicsProgram



# Our First GraphicsProgram

```
import acm.program.*;
import acm.graphics.*; // Stanford graphical objects
import jawa.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

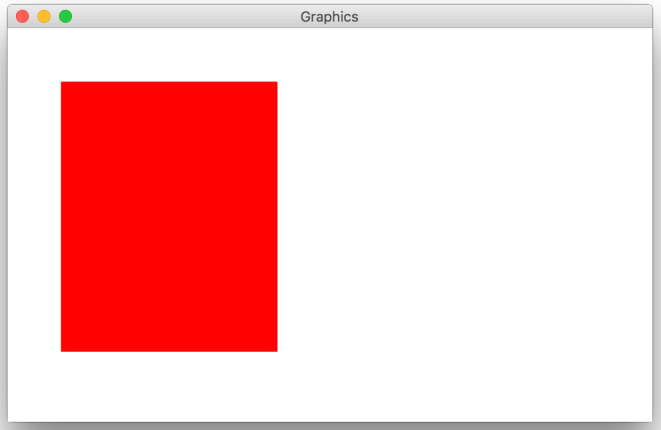
```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

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

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

# Our First GraphicsProgram

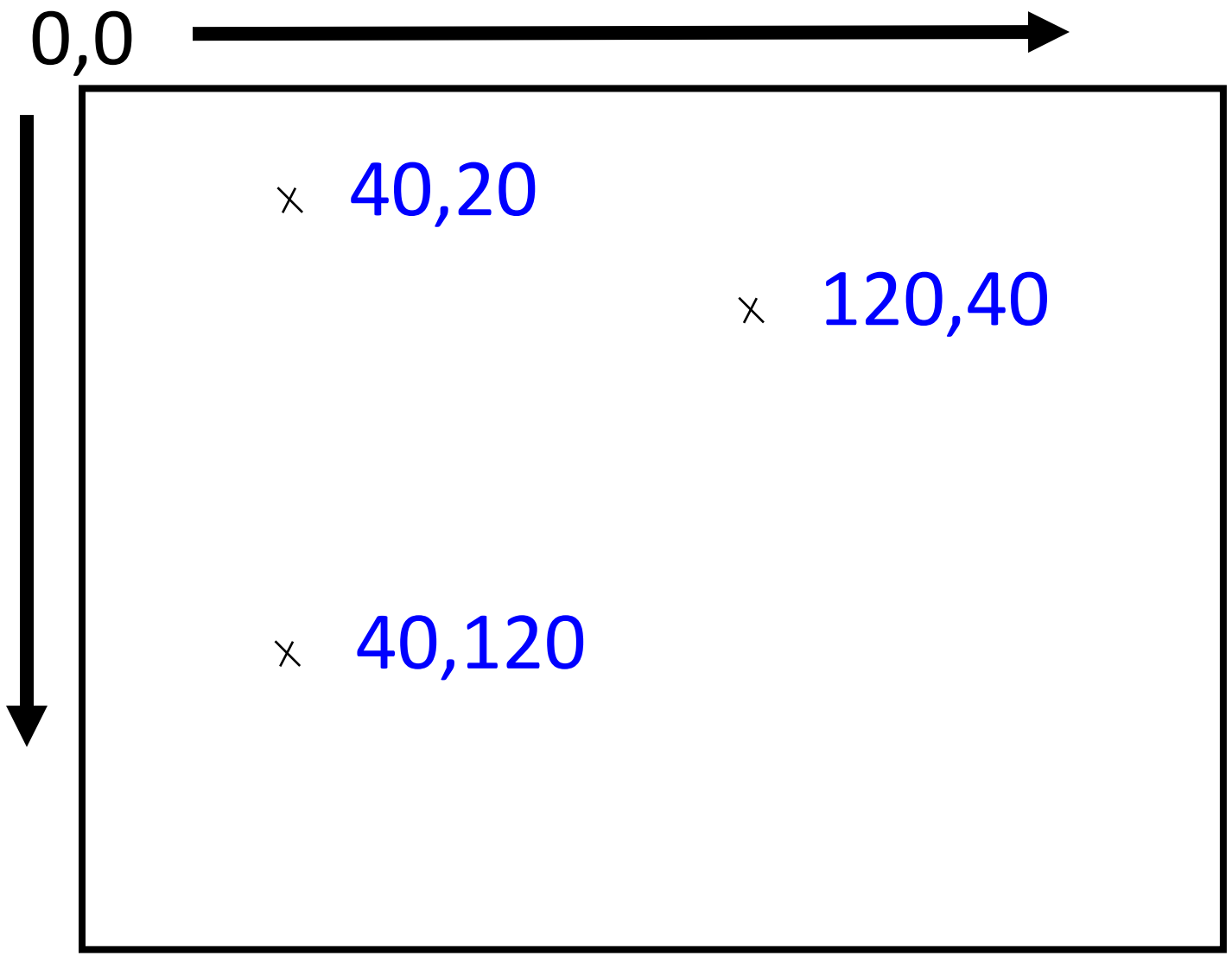
```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

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

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

# Our First GraphicsProgram

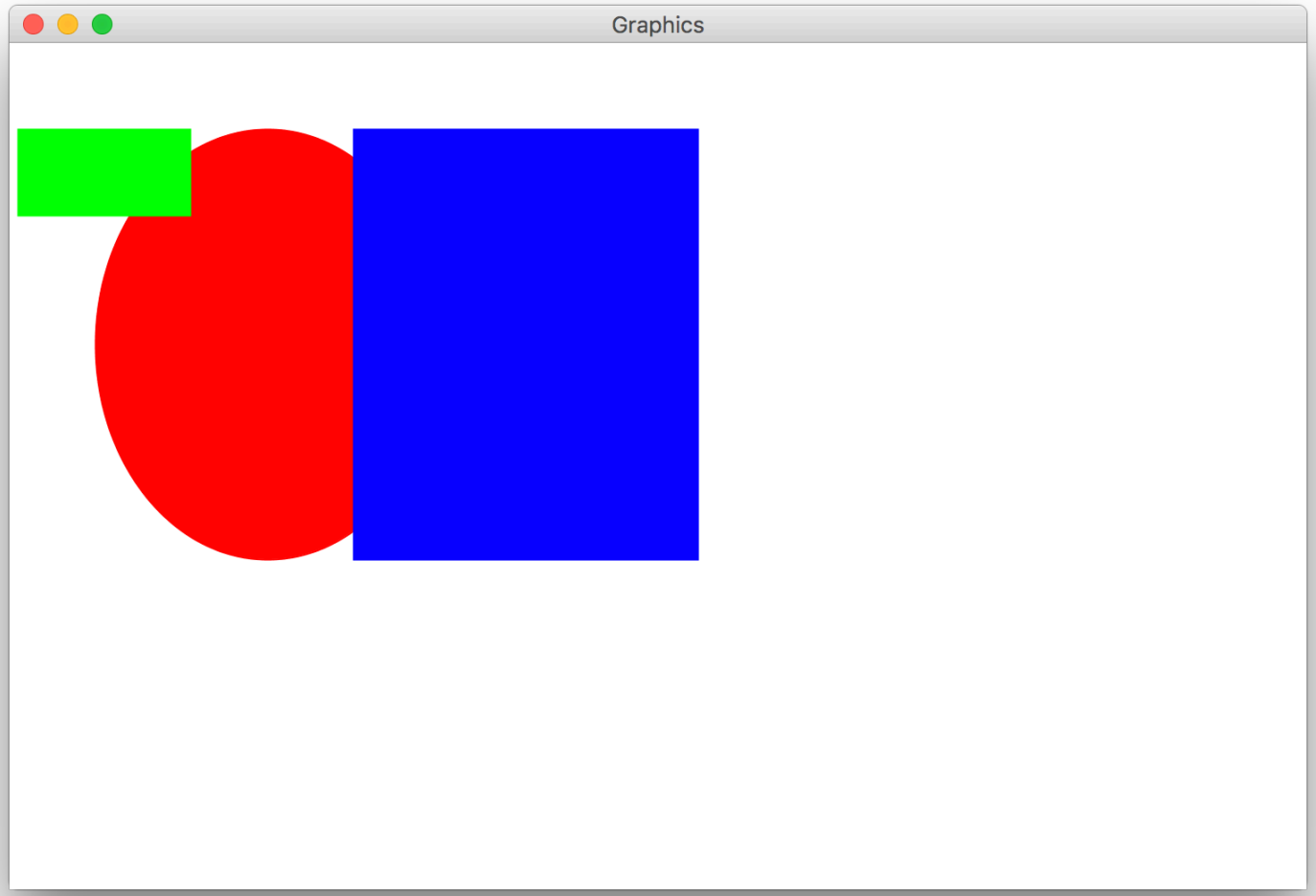
```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

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

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



# The Graphics Canvas



# Collage Model



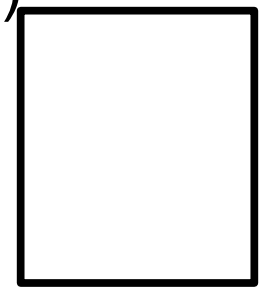
# Plan For Today

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

# Graphical Objects

`GRect`

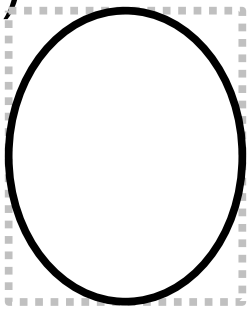
(x, y)



(x+w, y+h)

`GOval`

(x, y)



(x+w, y+h)

`GLine`

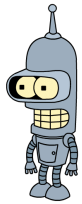
$(x_1, y_1)$

$(x_2, y_2)$

`GLabel`

*Hello there!*

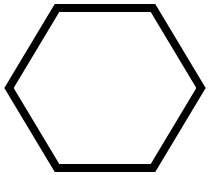
`GImage`



`GArc`

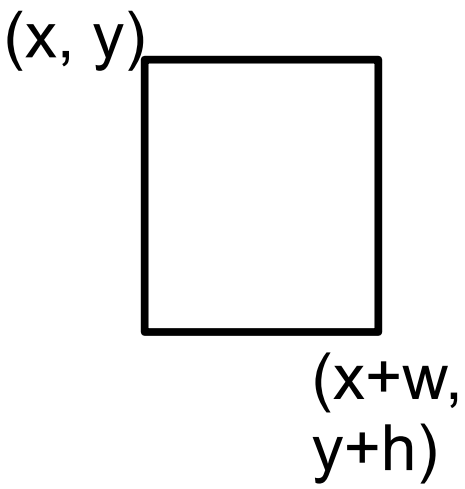
`GRoundRect`

`GPolygon`

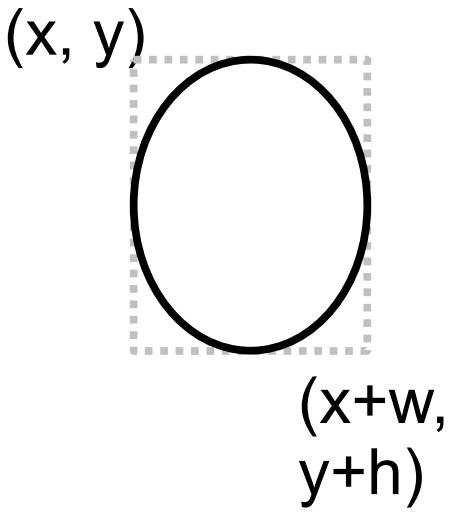


# Graphical Objects

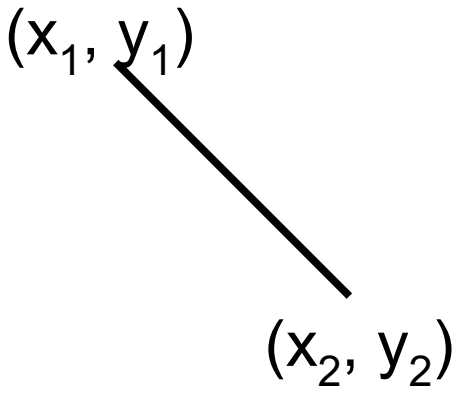
**GRect**




**GOval**



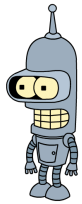

**GLine**




**GLabel**

*Hello there!*

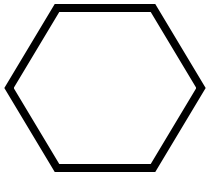
**GImage**



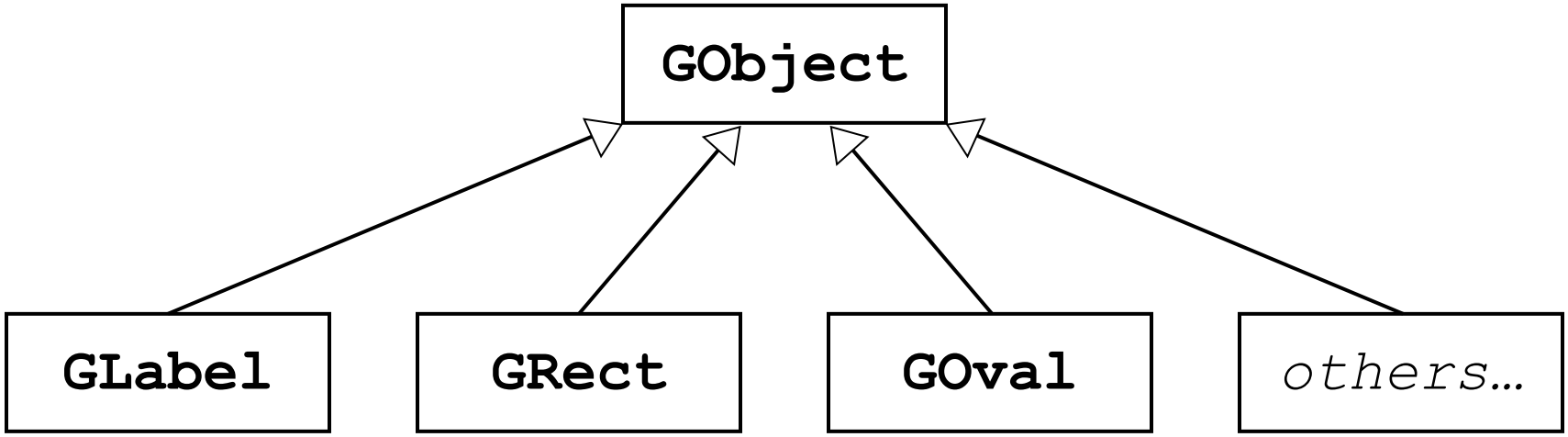
GArc

GRoundRect

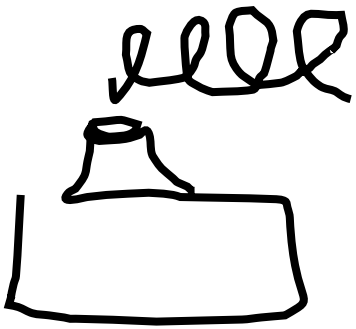
GPolygon



# Graphical Objects




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



# Primitives vs. Objects

Primitive Variable Types

```
int
double
char
boolean
```

Object Variable Types

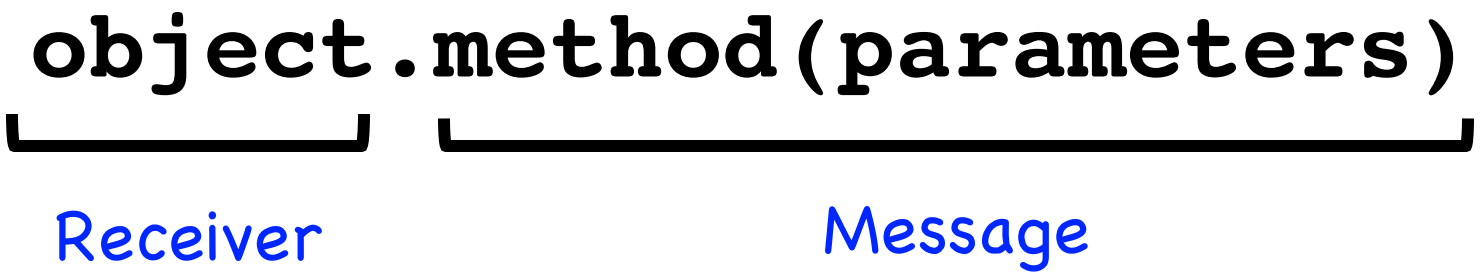
```
GRect
GOval
GLine
Scanner
...
```

Object variables:

1. Have upper camel case types
2. You can call methods on them
3. Are constructed using `new`

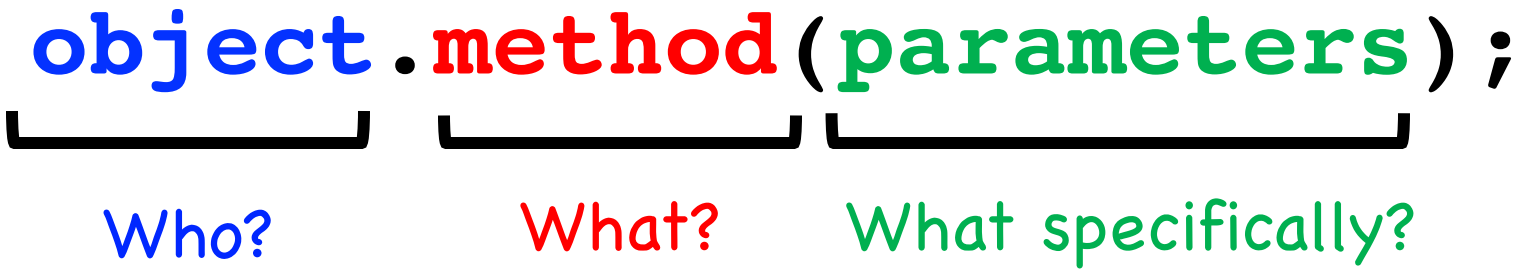
# Methods on Graphics Objects

We manipulate graphics objects by calling methods on them:



# Methods on Graphics Objects

We manipulate graphics objects by calling methods on them:




**Example:**

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

# GObject Methods

The following operations apply to all **`GObject`**s:

| |
|---|
| *object*`.`**`setColor`**`(`*color*`)`<br>Sets the color of the object to the specified color constant. |
| *object*`.`**`setLocation`**`(`*x*`, `*y*`)`<br>Changes the location of the object to the point (*x*, *y*). |
| *object*`.`**`move`**`(`*dx*`, `*dy*`)`<br>Moves the object on the screen by adding *dx* and *dy* to its current coordinates. |
| *object*`.`**`getWidth`**`()`<br>Returns the width of the object |
| *object*`.`**`getHeight`**`()`<br>Returns the height of the object |

*Graphic courtesy of Eric Roberts*

# 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));
```

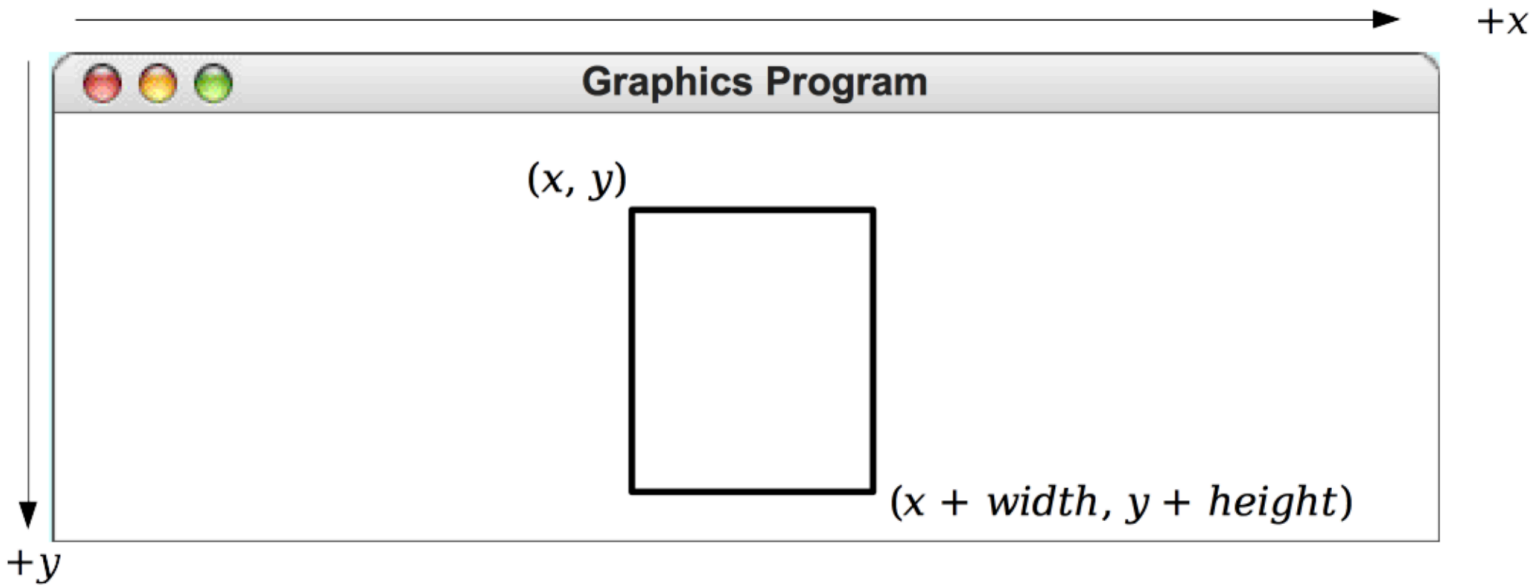
# GRect

**`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)



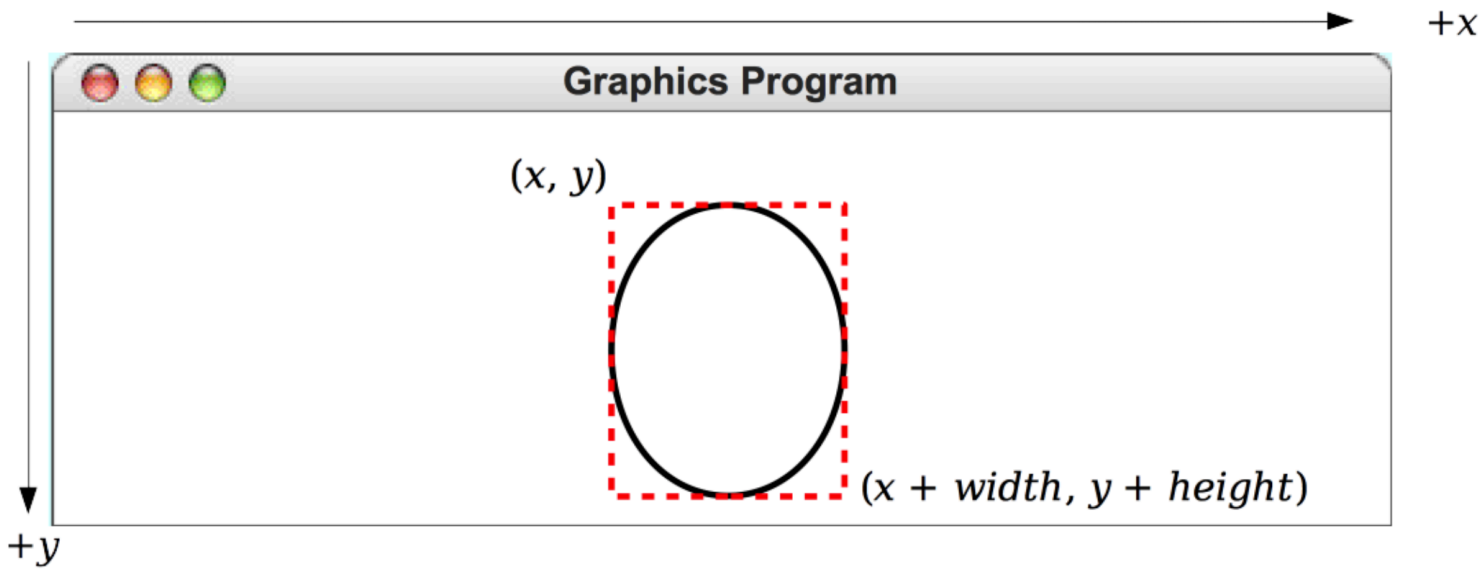
# GOval

**`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

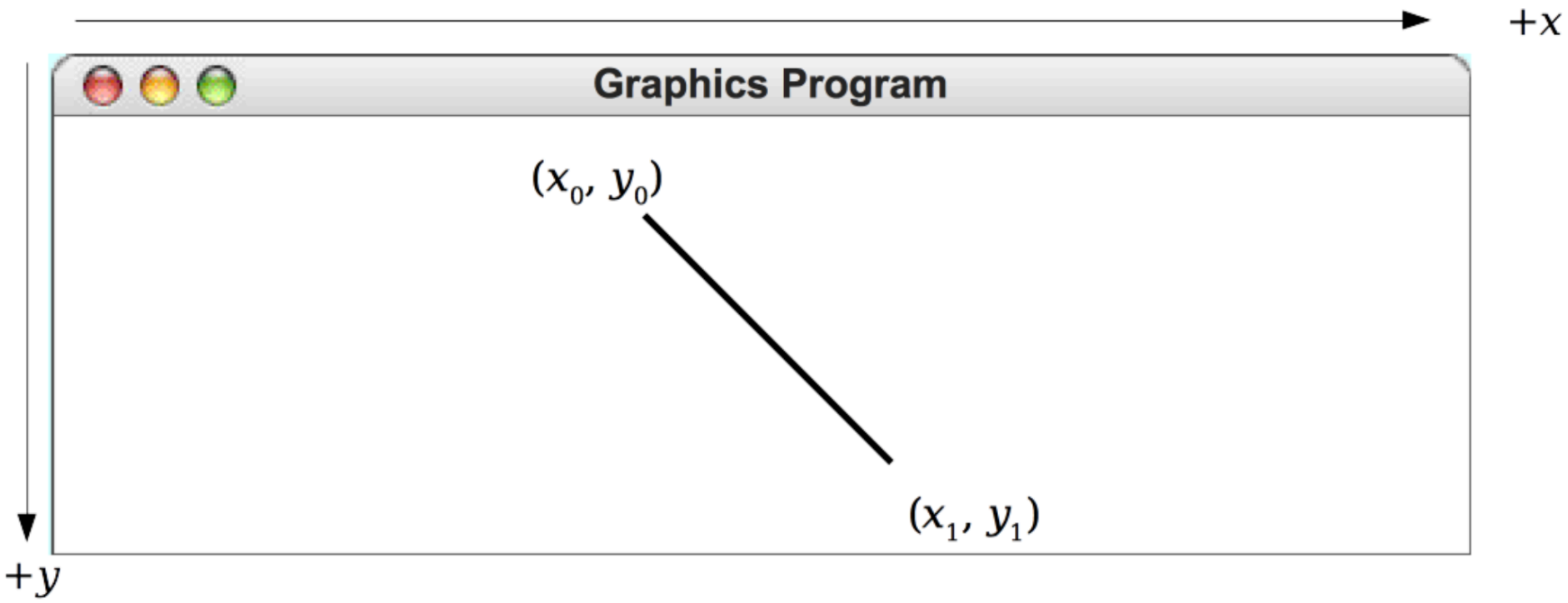
Methods shared by the **`GRect`** and **`GOval`** classes

| |
|---|
| *object*`.`**`setFilled(`***fill***`)`**<br>If *fill* is **`true`**, fills in the interior of the object; if **`false`**, shows only the outline. |
| *object*`.`**`setFillColor(`***color***`)`**<br>Sets the color used to fill the interior, which can be different from the border. |
| *object*`.`**`setSize(`***width, height***`)`**<br>Sets the object's size to be the given width and height |

# GLine

**`new GLine(`*`x0, y0, x1, y1`*`);`**

- Creates a line extending from (x0, y0) to (x1, y1)



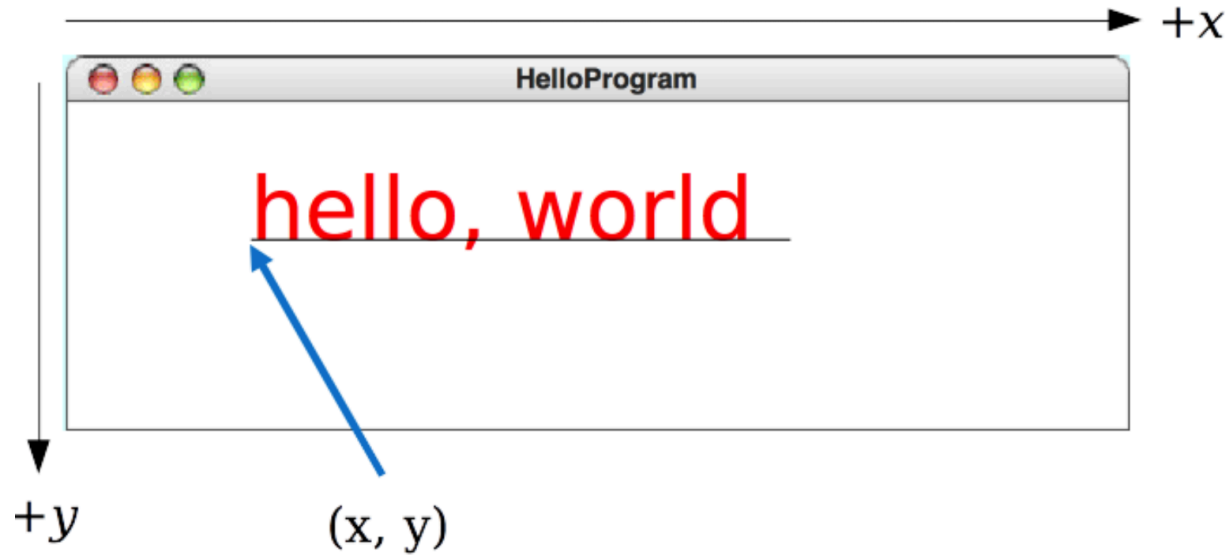
# GLabel

**`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()`**<br>Returns the height of the label below its baseline. |
| *label*`.`**`getAscent()`**<br>Returns the height of the label above its baseline. |
| *label*`.`**`setFont(`***font***`)`**<br>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

*Graphic courtesy of Eric Roberts*

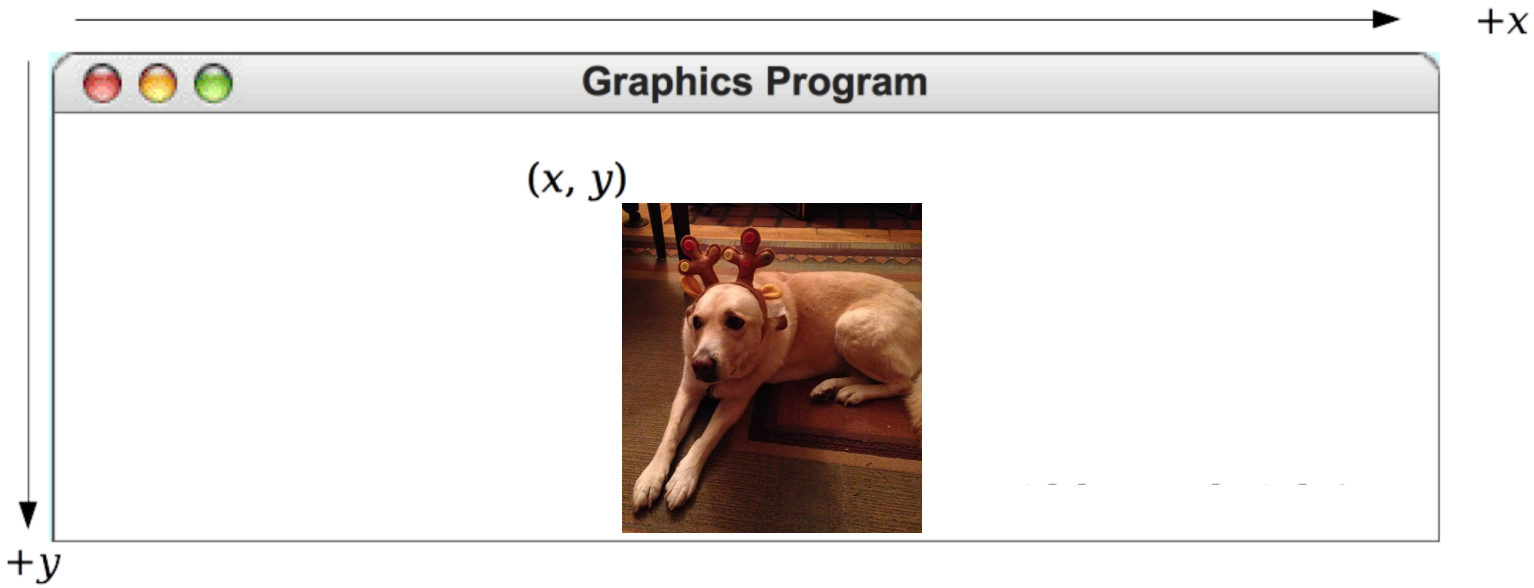
# GImage

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

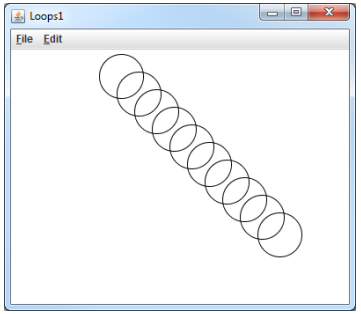
# Plan For Today

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

# Practice: Drawing w/ Loops

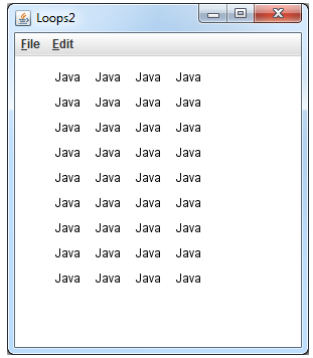
- The *x*,*y*,*w*,*h* expressions can use the loop counter variable:

```
for (int i = 0; i < 10; i++) {
    add(new GOval(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));
    }
}
```

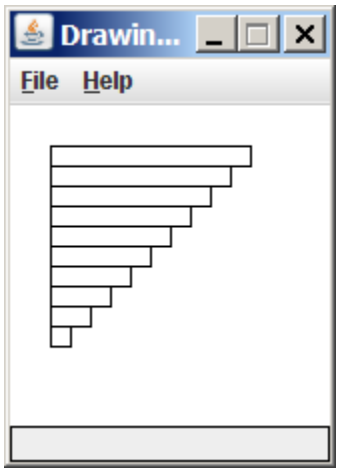


# Practice: Drawing w/ Loops

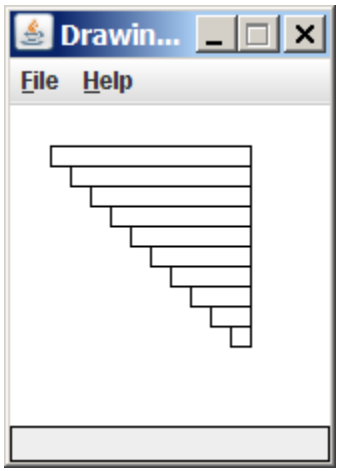
- **Q:** What is the output of the following code?

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

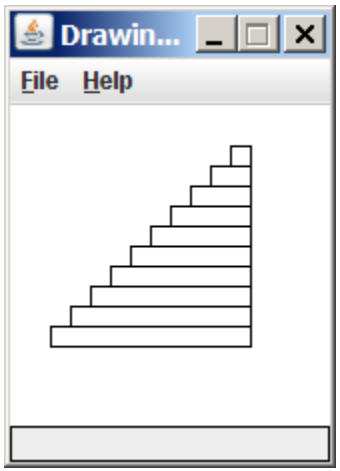
**A.**




**B.**




**C.**




**D.**

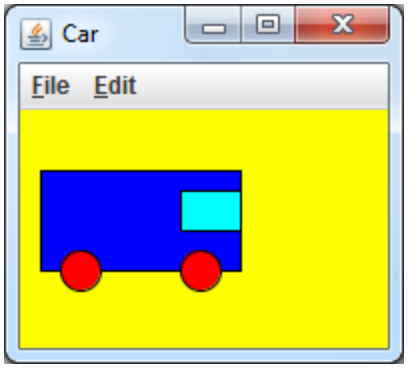
none

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

# Practice: Car

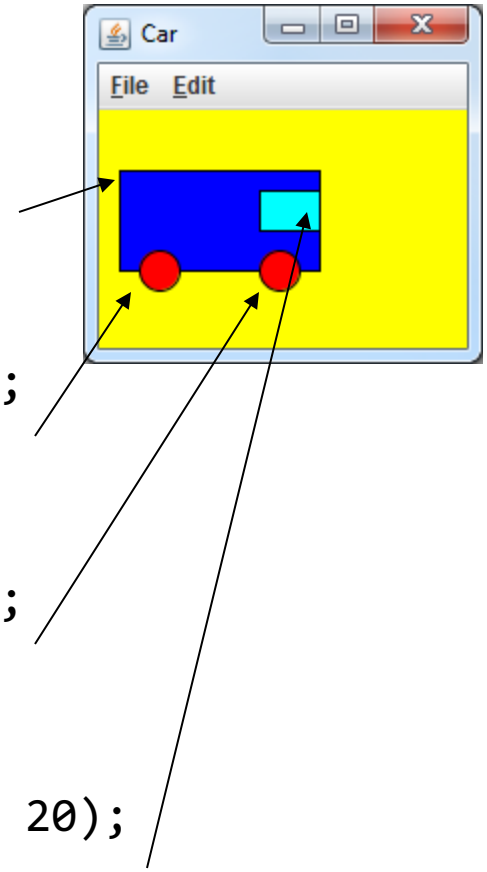
Write a graphical program named **Car** that draws a figure that looks (kind of) like a car.

- 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



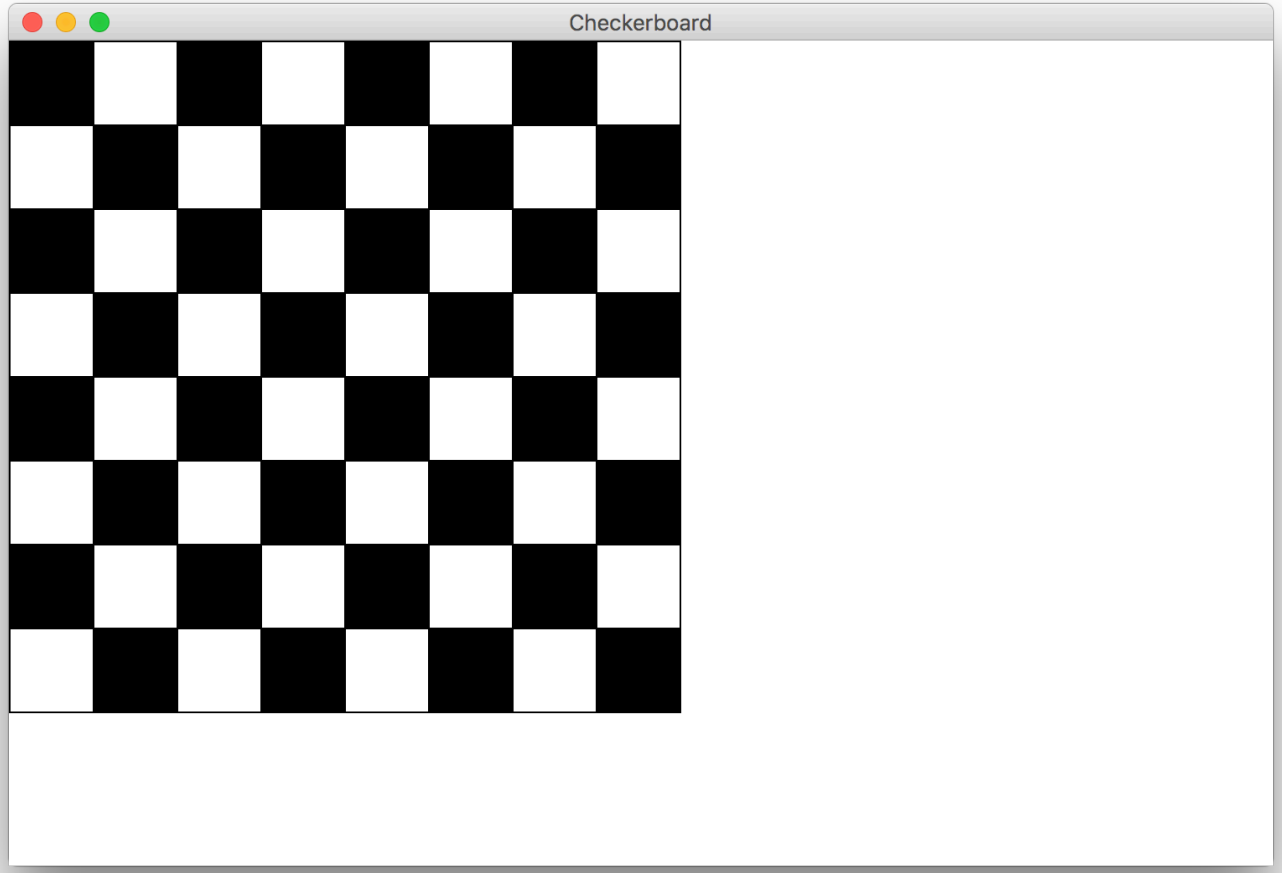
# Car Solution

```
// When 2 shapes occupy the same pixels, the last one drawn "wins"
public class Car extends GraphicsProgram {
    public void run() {
        setBackground(Color.YELLOW);

        GRect body = new GRect(10, 30, 100, 50);
        body.setFilled(true);
        body.setFillColor(Color.BLUE);
        add(body);

        GOval wheel1 = new GOval(20, 70, 20, 20);
        wheel1.setFilled(true);
        wheel1.setFillColor(Color.RED);
        add(wheel1);

        GOval wheel2 = new GOval(80, 70, 20, 20);
        wheel2.setFilled(true);
        wheel2.setFillColor(Color.RED);
        add(wheel2);

        GRect windshield = new GRect(80, 40, 30, 20);
        windshield.setFilled(true);
        windshield.setFillColor(Color.CYAN);
        add(windshield);
    }
}
```

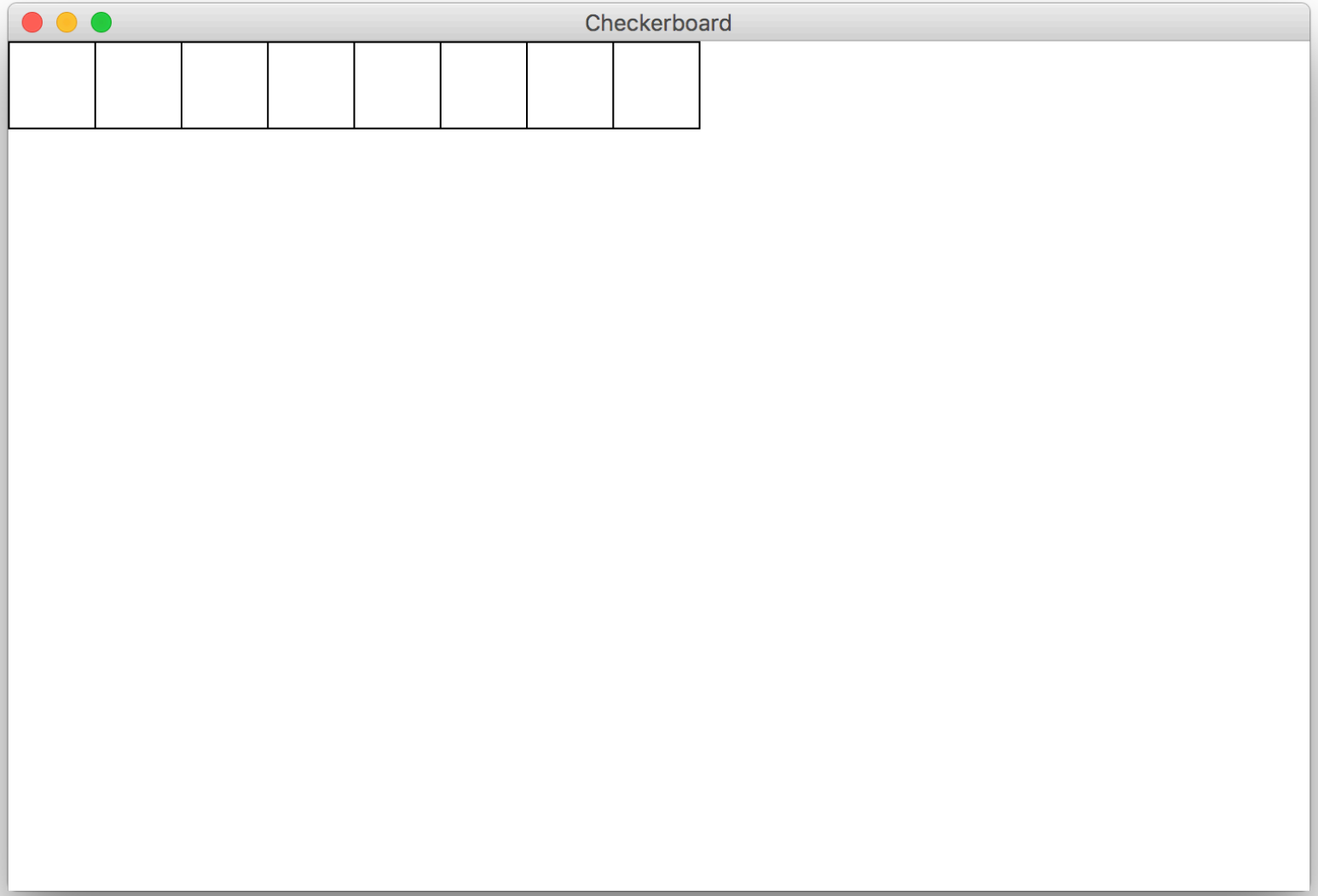


# Practice: Checkerboard

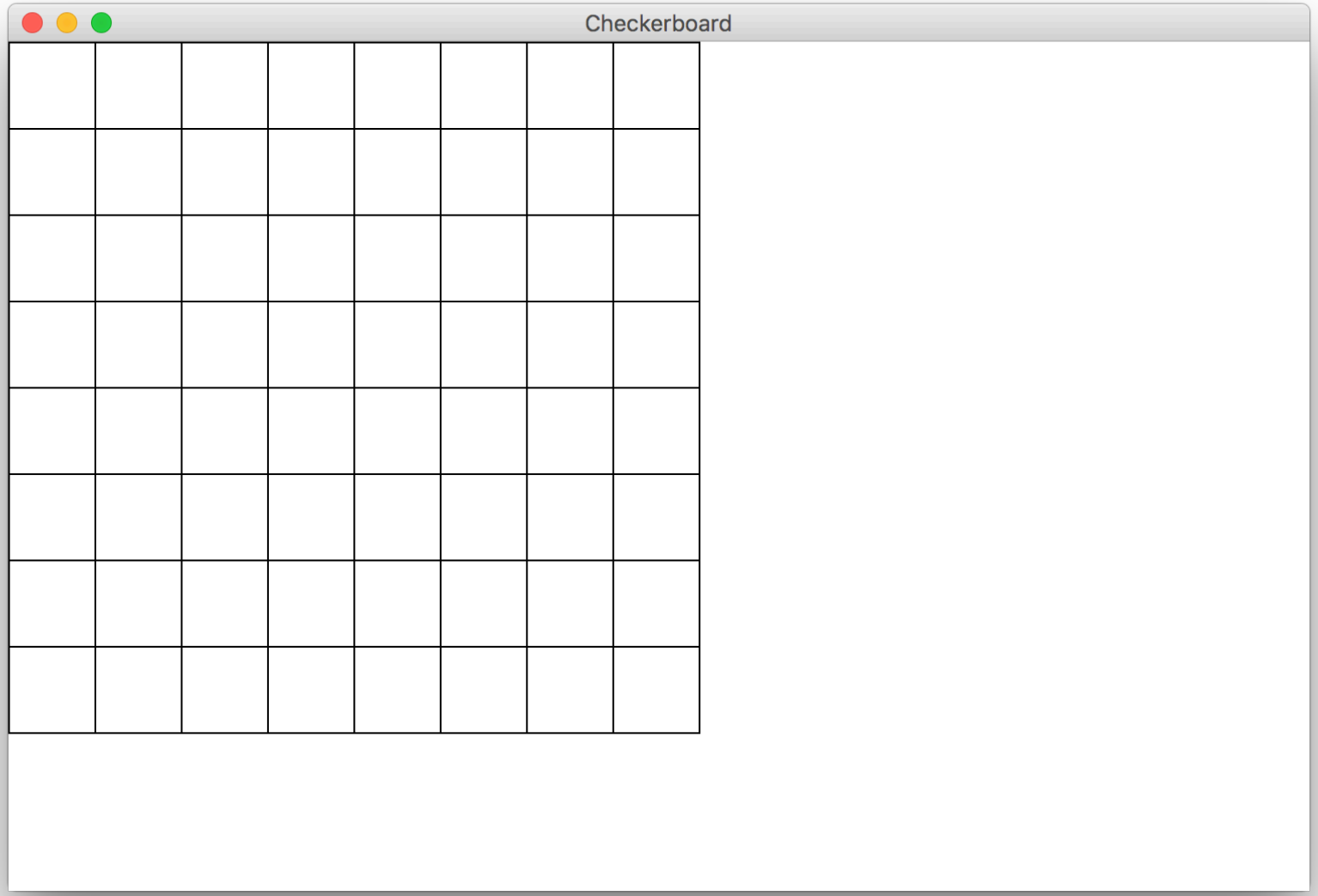
Write a graphical program named **`Checkerboard`** that draws a checkerboard pattern using GRects.



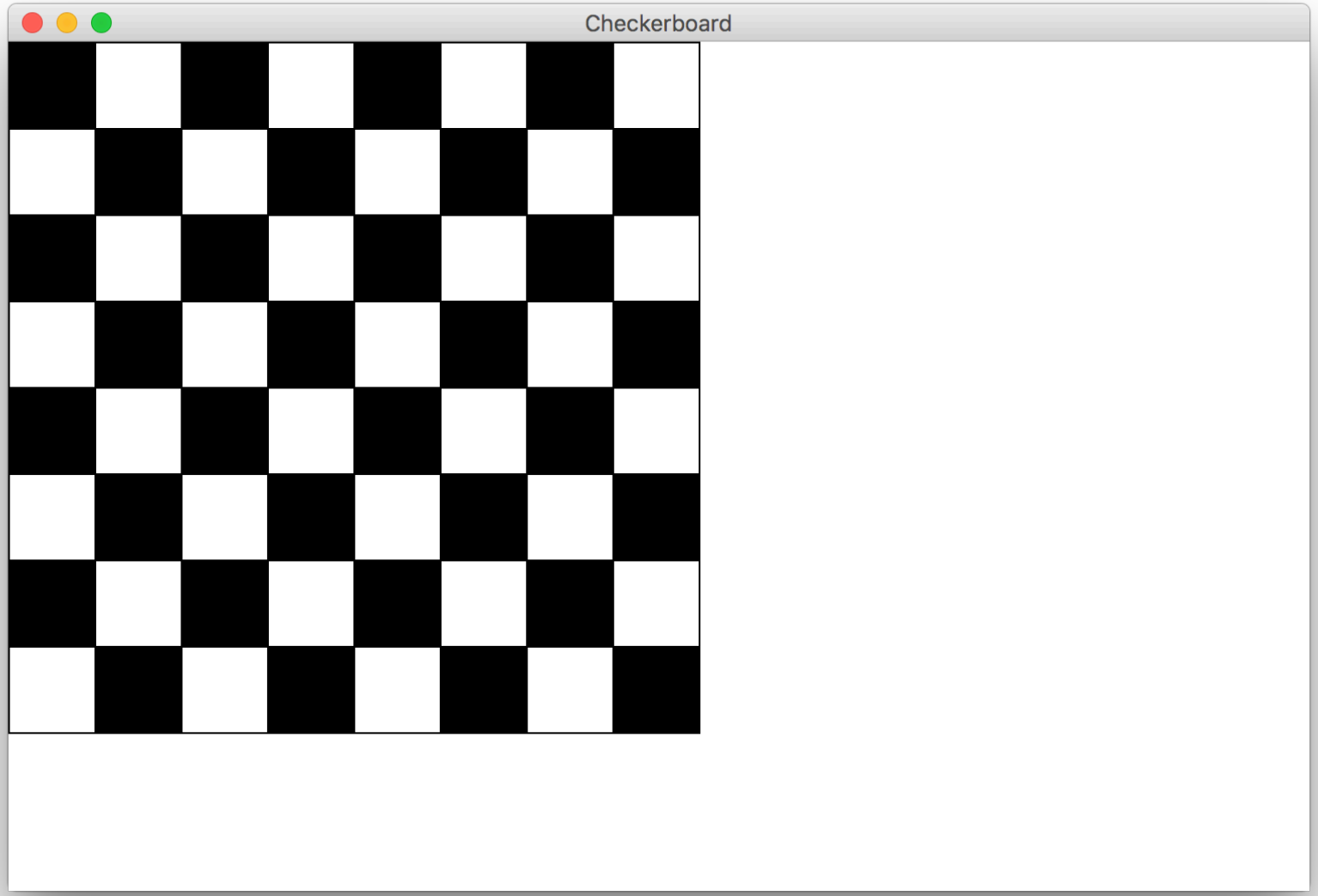
# Milestone 1



# Milestone 2

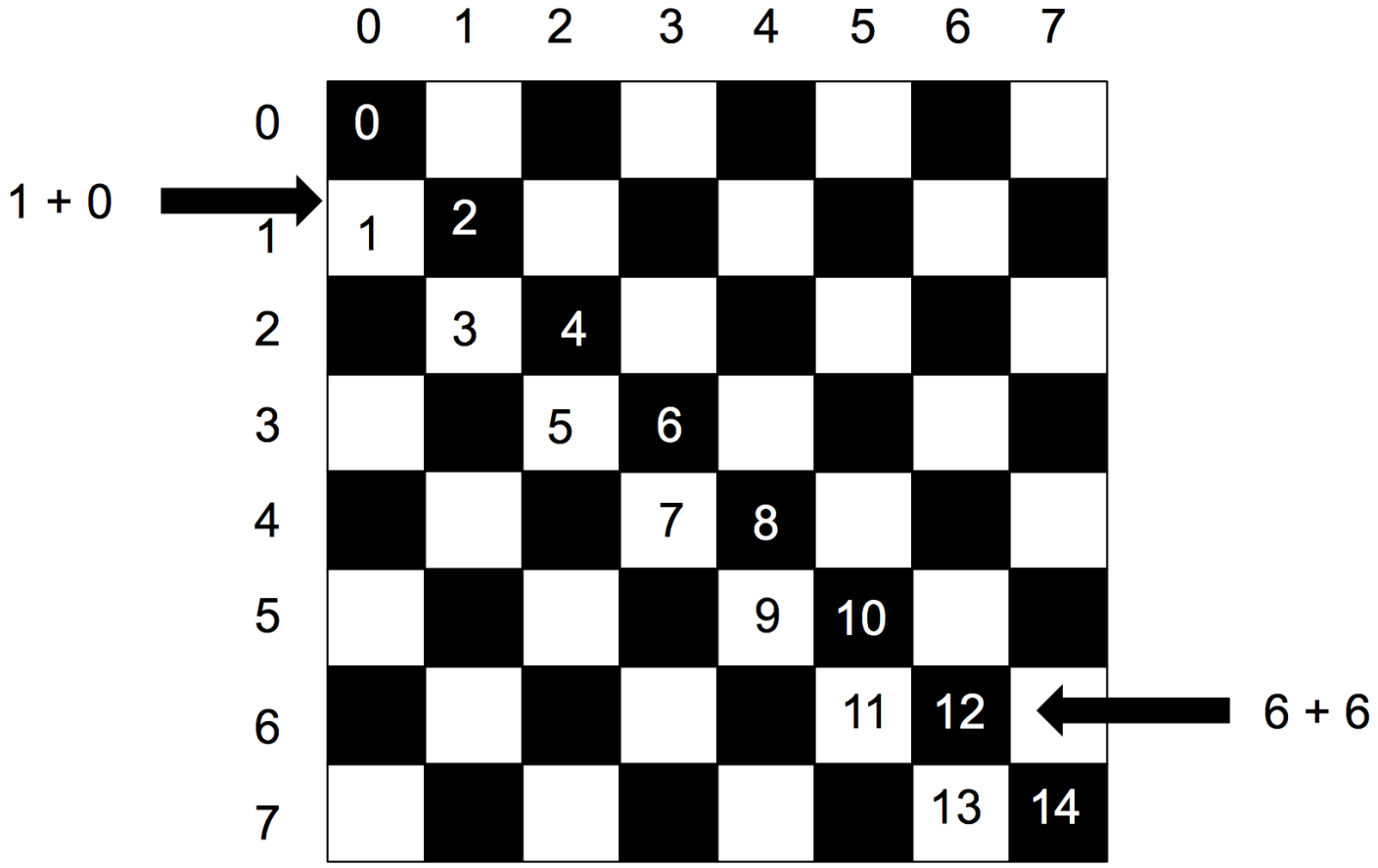


# Milestone 3



# Milestone 3

- Notice the pattern if we add the row and column indexes…



# Recap

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

**Next time: Animation**