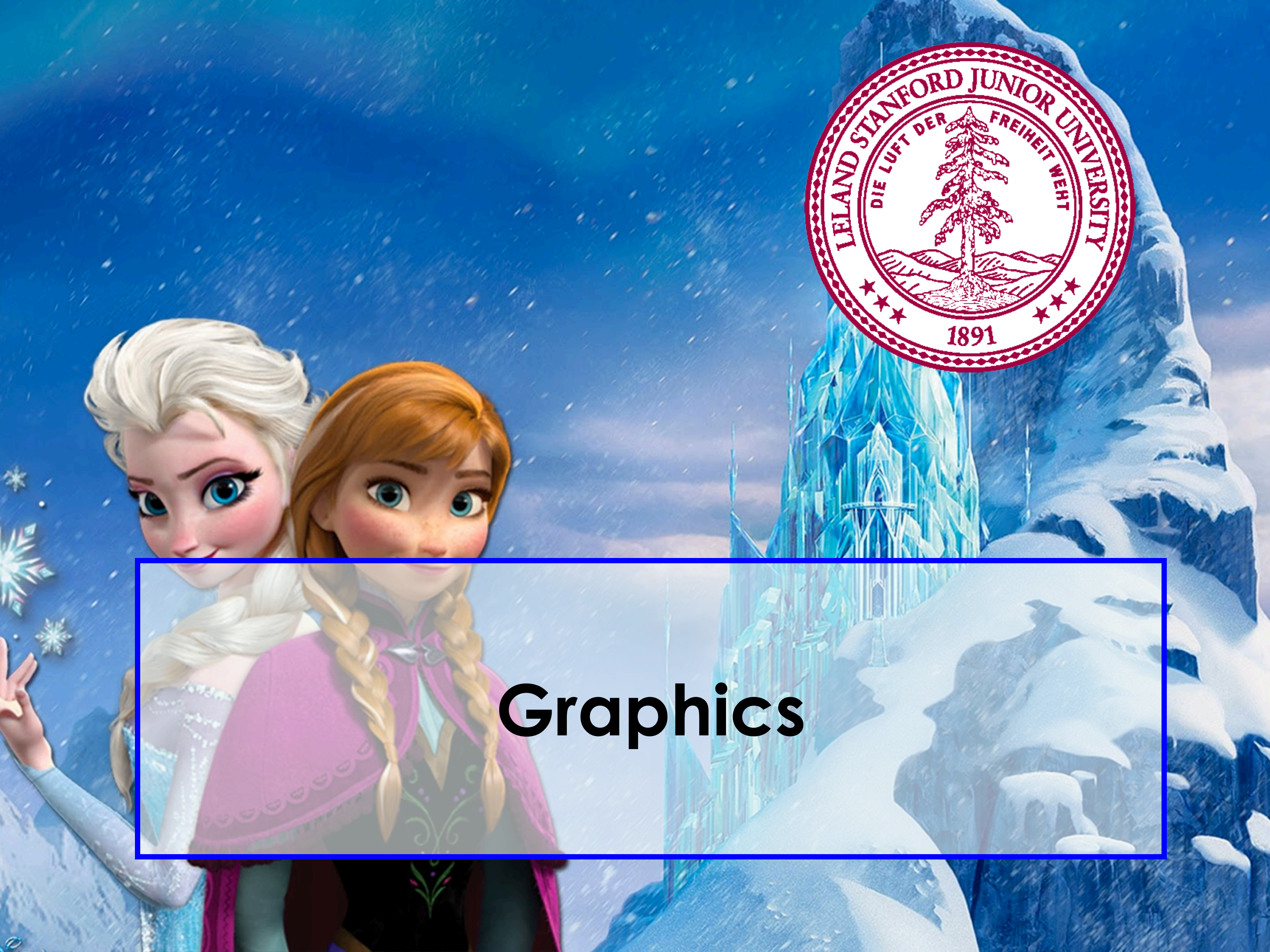
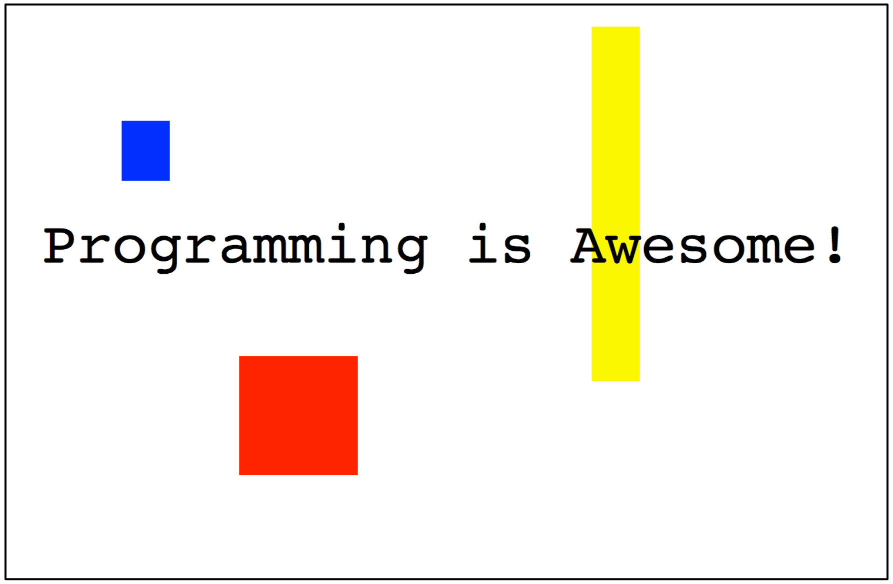


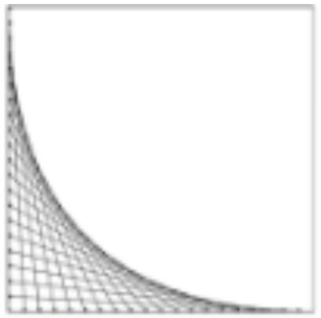
Graphics

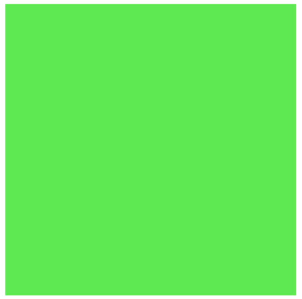
Plan For Today

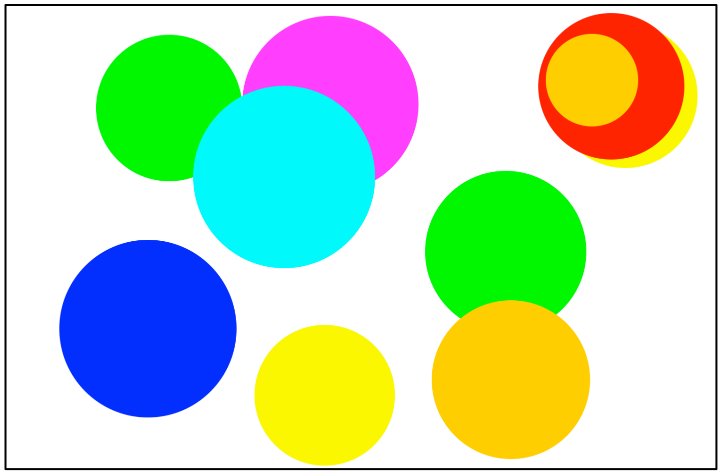
- Day 3 Overview
- Recap: Introduction to Java
- GraphicsProgram
- Graphical Objects
- Practice: Car

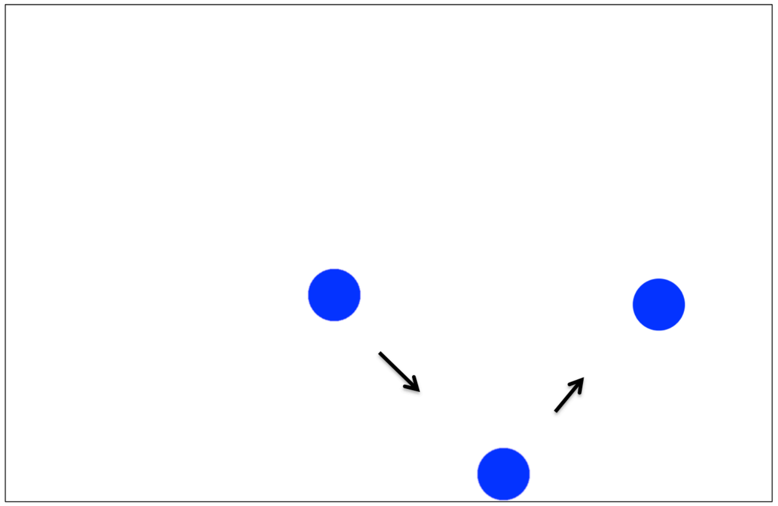
Day 3











Plan For Today

- Day 3 Overview
- **Recap: Introduction to Java**
- GraphicsProgram
- Graphical Objects
- Practice: Car

Variable Declaraion

```
// Declares an int
int total = 42;


// Changes its value
total = 4;
```

Types

```
String str = “Hi”;

int num = 5;

double fraction = 0.2
```

Expressions

- You can combine literals or variables together into **expressions** using binary operators:

+	Addition	*	Multiplication
–	Subtraction	/	Division
		%	Remainder

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `*` `/` `%` have a higher level of precedence than `+` `-`

```
1 + 3 * 4         is 13

6 + 8 / 2 * 3
6 +   4   * 3
6 +     12        is 18
```

  - Parentheses can alter order of evaluation, but spacing does not:

```
(1 + 3) * 4       is 16
1+3 * 4-2         is 11
```

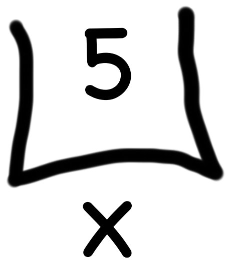
Lifetime of a Variable

A Variable love story

Once upon a time…

…x was looking for love!

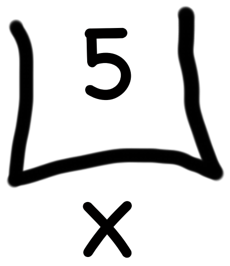
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



...x was looking for love!

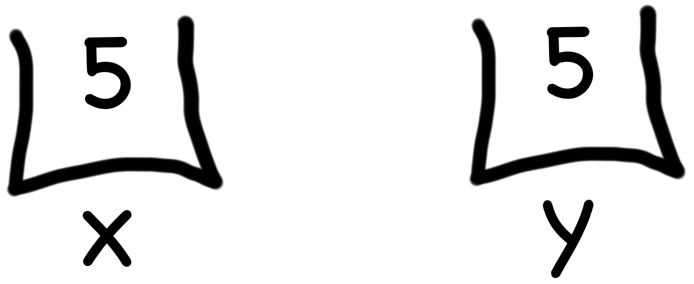
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

x was definitely looking for love



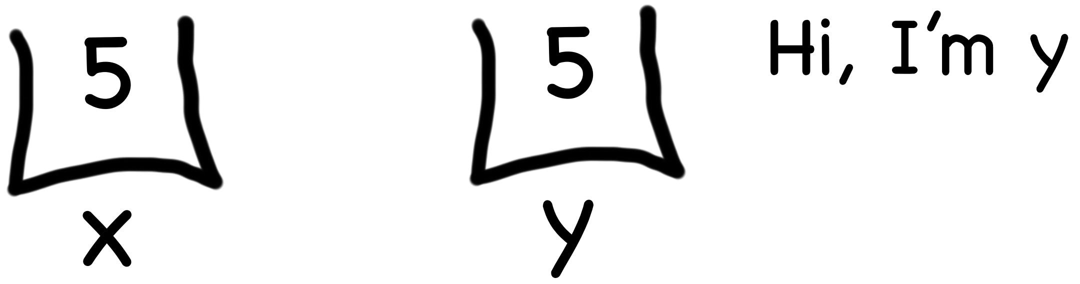
And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



And met y.

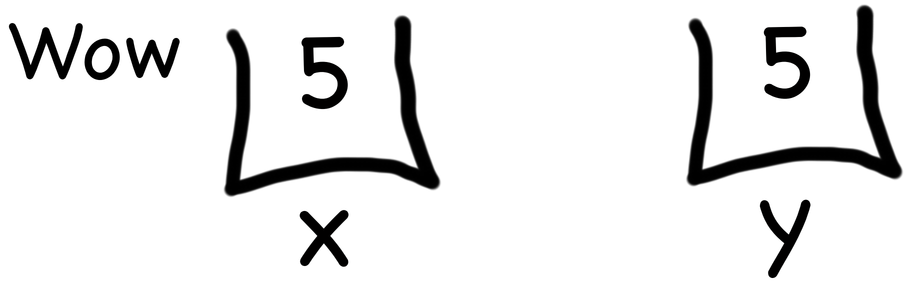
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



“Wow!”

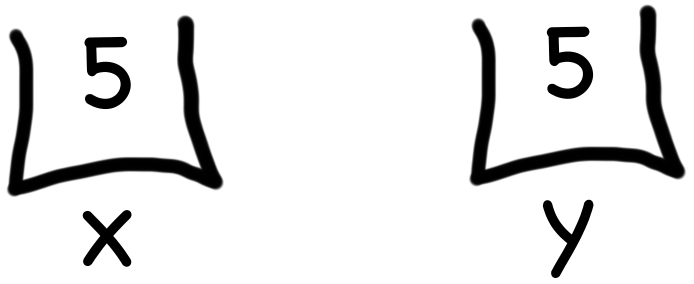
And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



And met y.

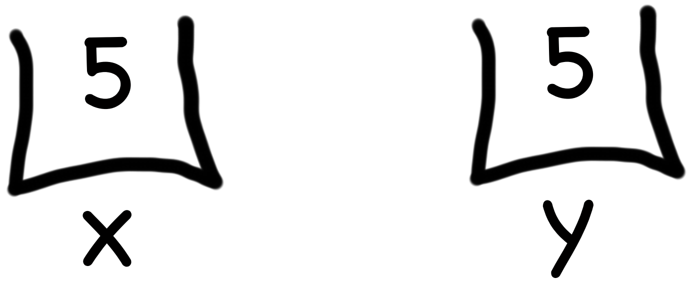
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



We have so much in common

And met y.

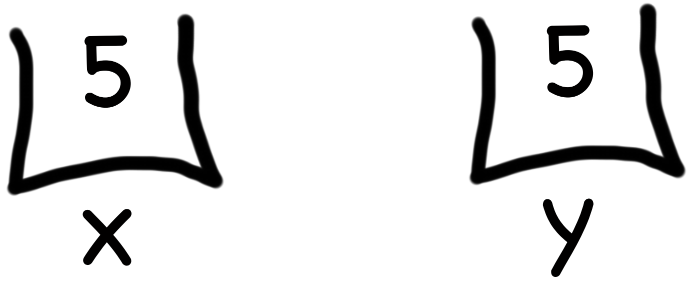
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



We both have value 5!

And met y.

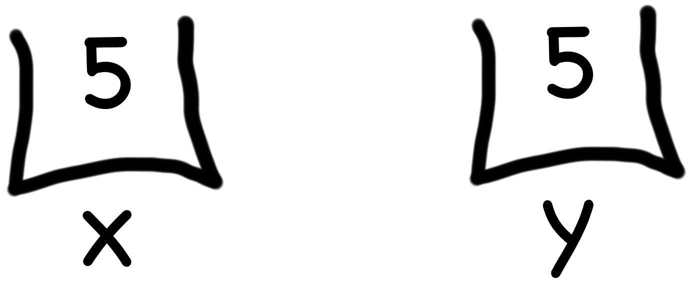
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



Maybe sometime we can...

And met y.

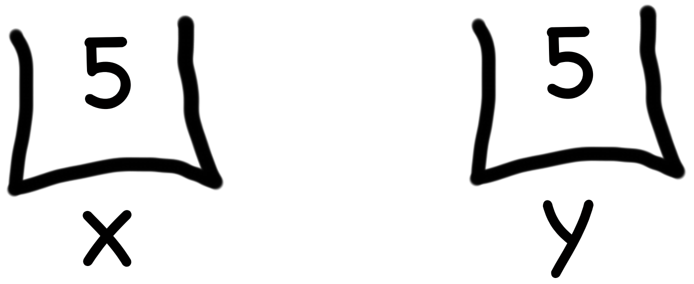
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



println together?

And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

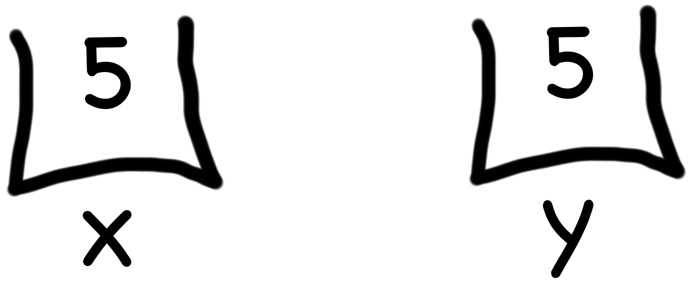


It was a beautiful match…

…but then tragedy struck.

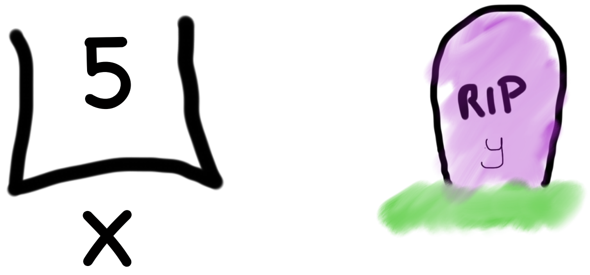
Tragedy Strikes

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



Tragedy Strikes

```
int x = 5;
if(lookingForLove()) {
   int y = 5;
}
println(x + y);
```

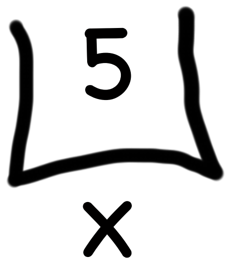


Noooooooooooooooo!

You see…
when a program exits a code block,
all variables declared inside that block go away!

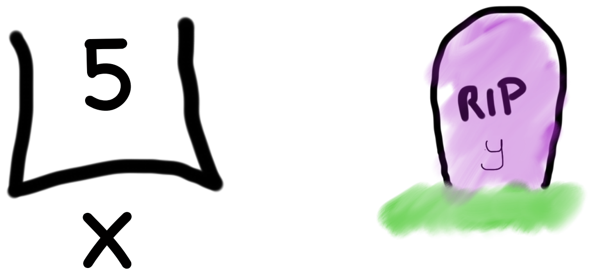
Since y is inside the if-block...

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



...it goes away here...

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```



...and doesn't exist here.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
}
println(x + y);
```

Error. Undefined variable y.

The End

Sad times ☹

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Variable Scope

Variables have a lifetime (called scope):

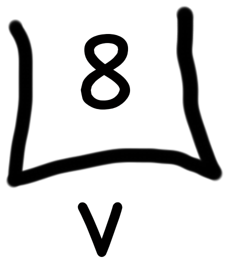
```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Comes to life here

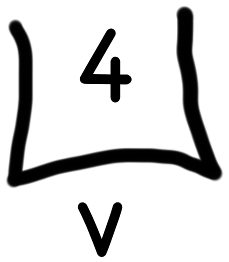


Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

This is the **inner most** code block in which it was declared….

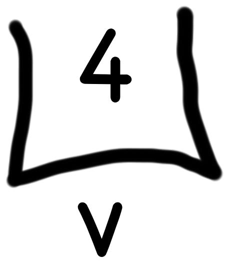


Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

Still alive here...

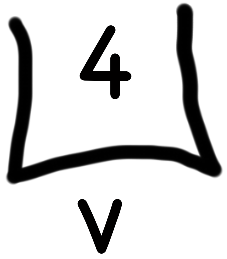


Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
   double v = 8;
   if(condition){
      v = 4;
      ... some code
   }
   ... some other code
}
```

It goes away here (at the end of its code block)



Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    double v = 8;
    if(condition){
        v = 4;
        ... some code
    }
    ... some other code
}
```

It goes away here (at the end of its code block)

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    ... some code
    if(condition){
        int w = 4;
        ... some code
    }
    ... some other code
}
```

This is the scope of **w**

Variable Scope

Variables have a lifetime (called scope):

```
public void run(){
    … some code
    if(condition){
        int w = 4;
        … some code
    }
    … some other code
}
```

w is created here

w goes away here (at the end of its code block)

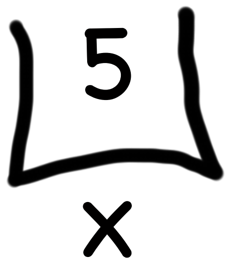
A Variable love story

Chapter 2

The programmer fixed the bug

...x was looking for love!

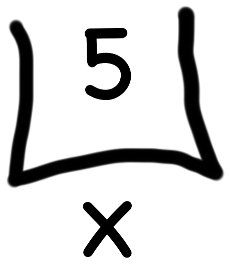
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```



…x was looking for love!

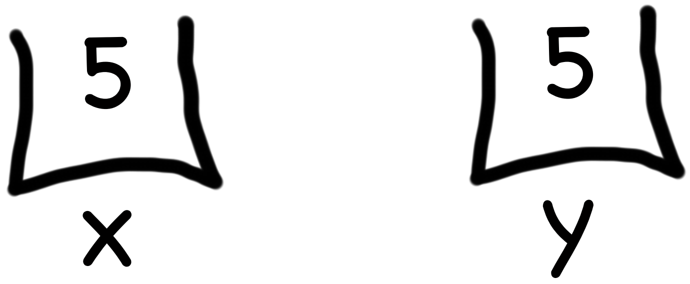
```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```

x was definitely looking for love



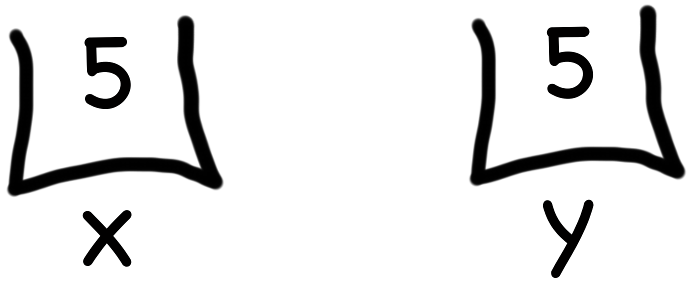
And met y.

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```



Since they were both “in scope”…

```
int x = 5;
if(lookingForLove()) {
    int y = 5;
    println(x + y);
}
```



…they lived happily ever after.
The end.

Variable Scope

- The **scope** of a variable refers to the section of code where a variable can be accessed.
- **Scope starts** where the variable is declared.
- **Scope ends** at the termination of the code block in which the variable was declared.

- A **code block** is a chunk of code between { } brackets

Variable Scope

You *cannot* have two variables with the same name in the *same scope*.

```
public void run() {
    int x = 5;
    ...
    int x = 2;                    // ERROR
}
```

Variable Scope

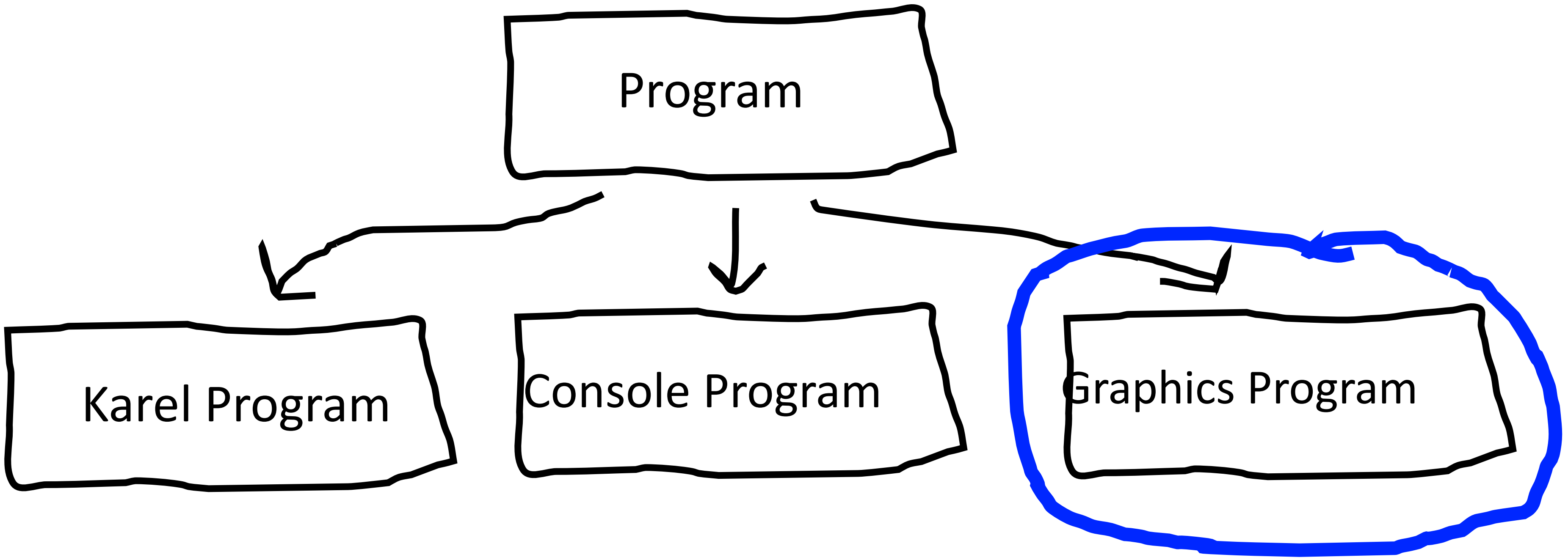
You *can* have two variables with the same name in *different scopes*.

```
public void run() {
    if (…) {
        int x = 5;
        println(x);
    } else {
        int x = 2;
        println(x);
    }
}
```

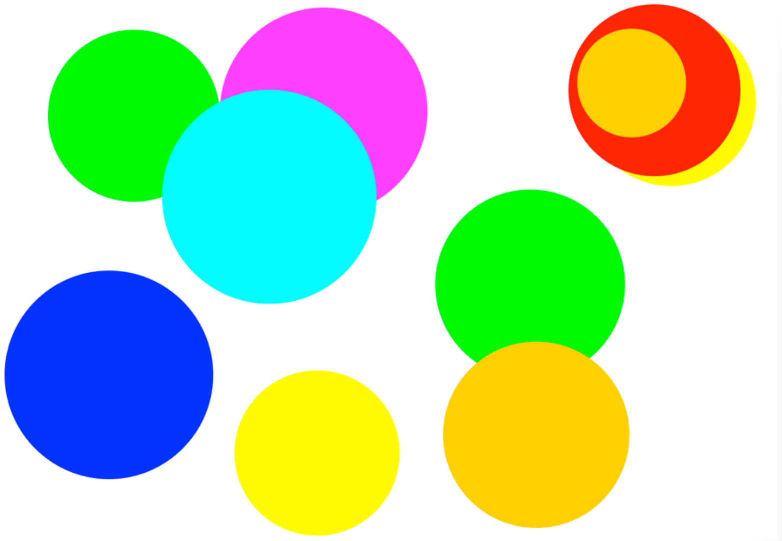
Plan For Today

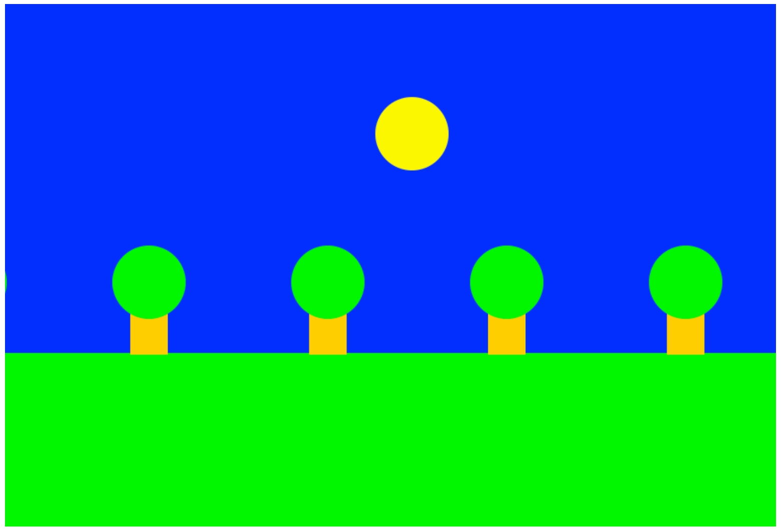
- Day 3 Overview
- Recap: Introduction to Java
- **GraphicsProgram**
- Graphical Objects
- Practice: Car

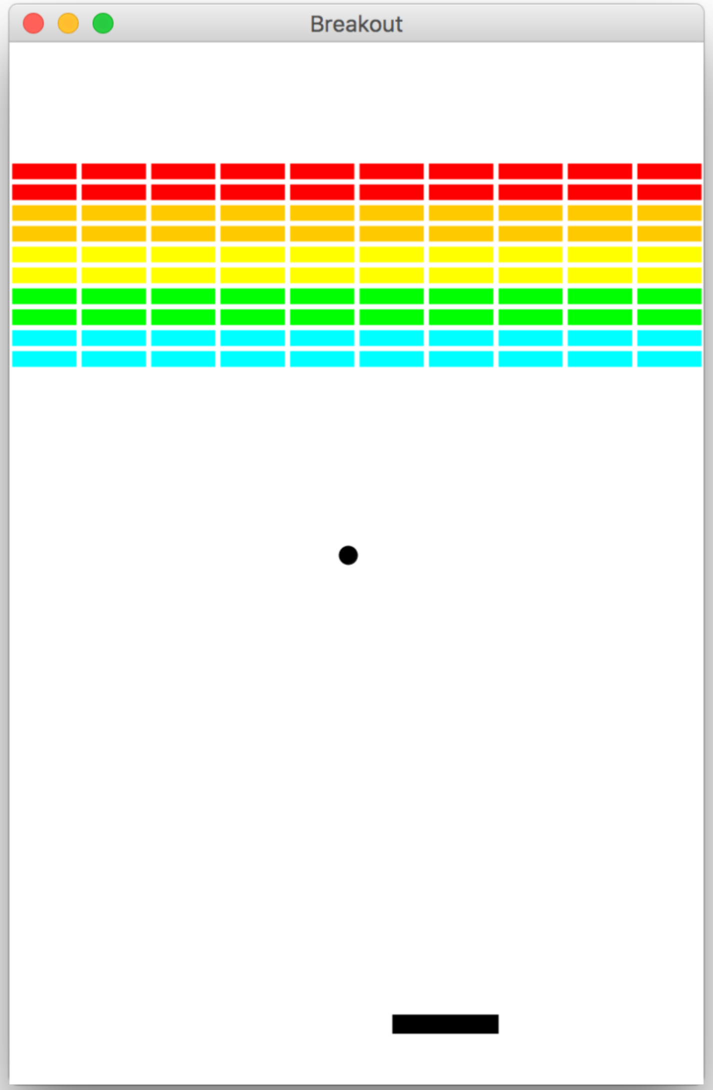
Java



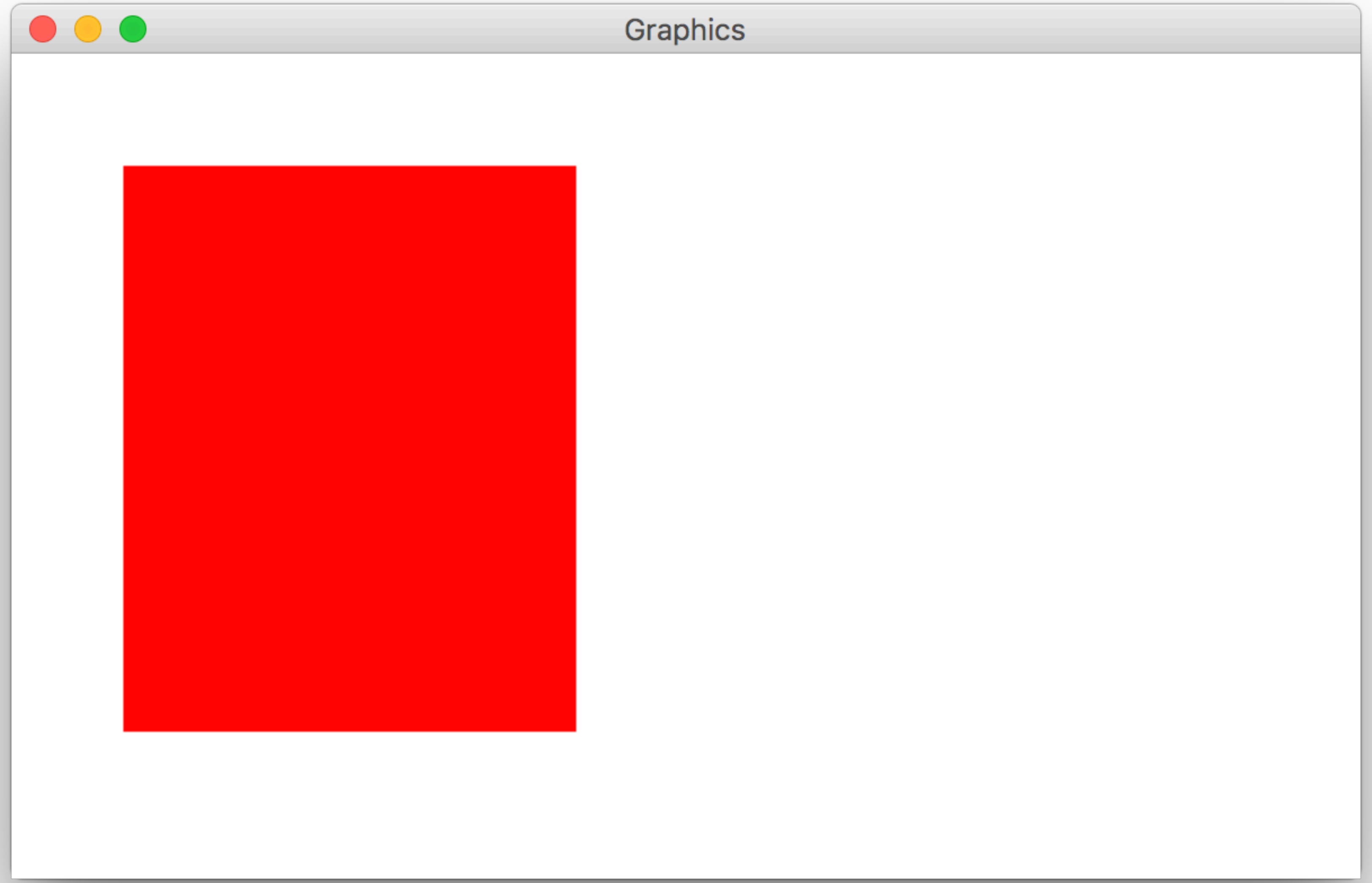
Graphics Programs







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

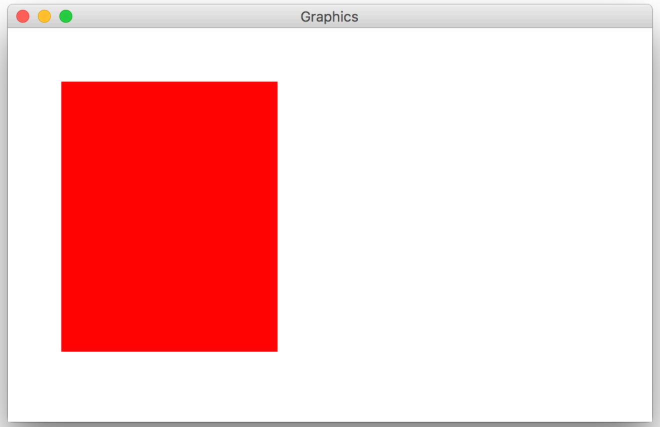
```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

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

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

Our First GraphicsProgram

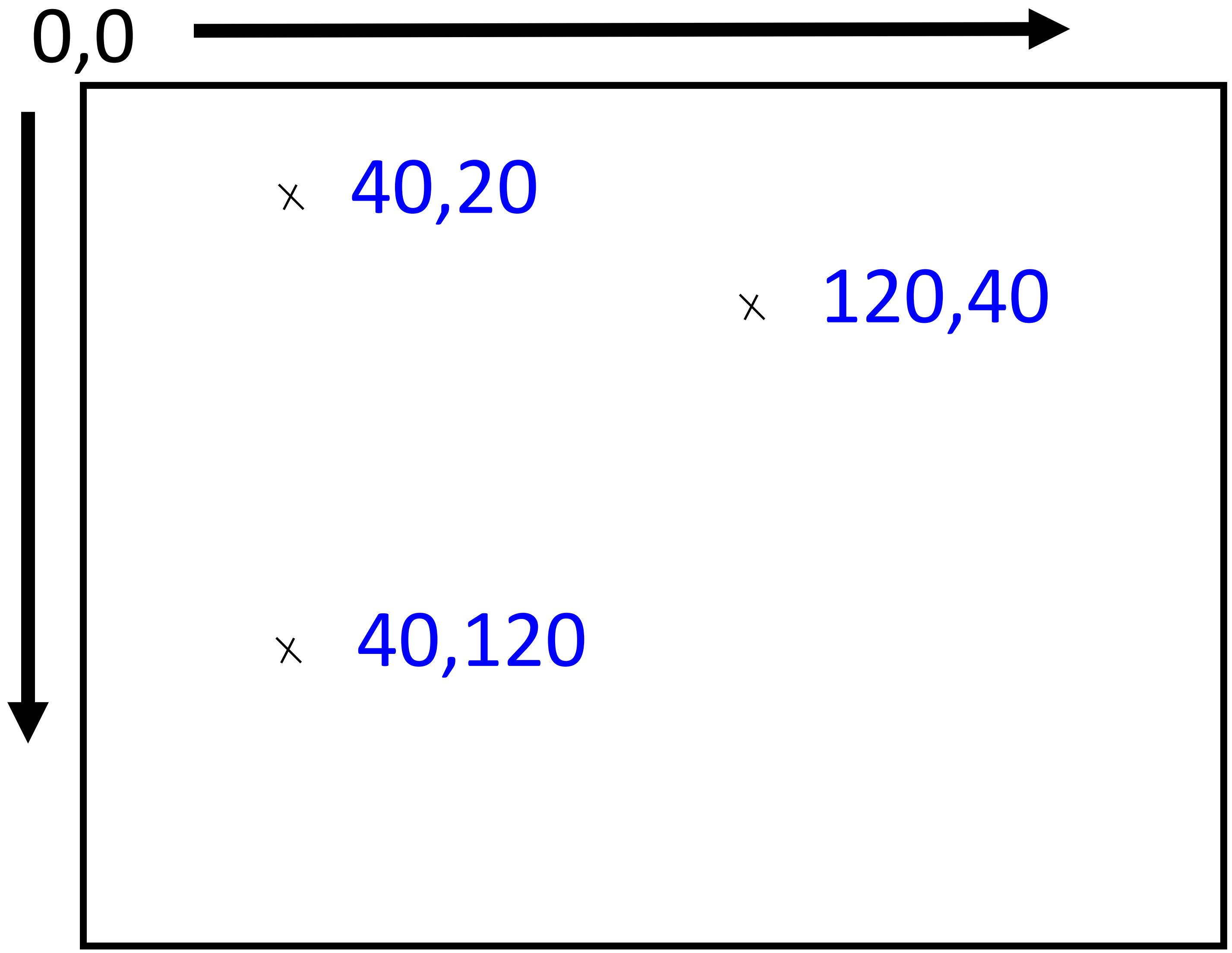
```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

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

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

Our First GraphicsProgram

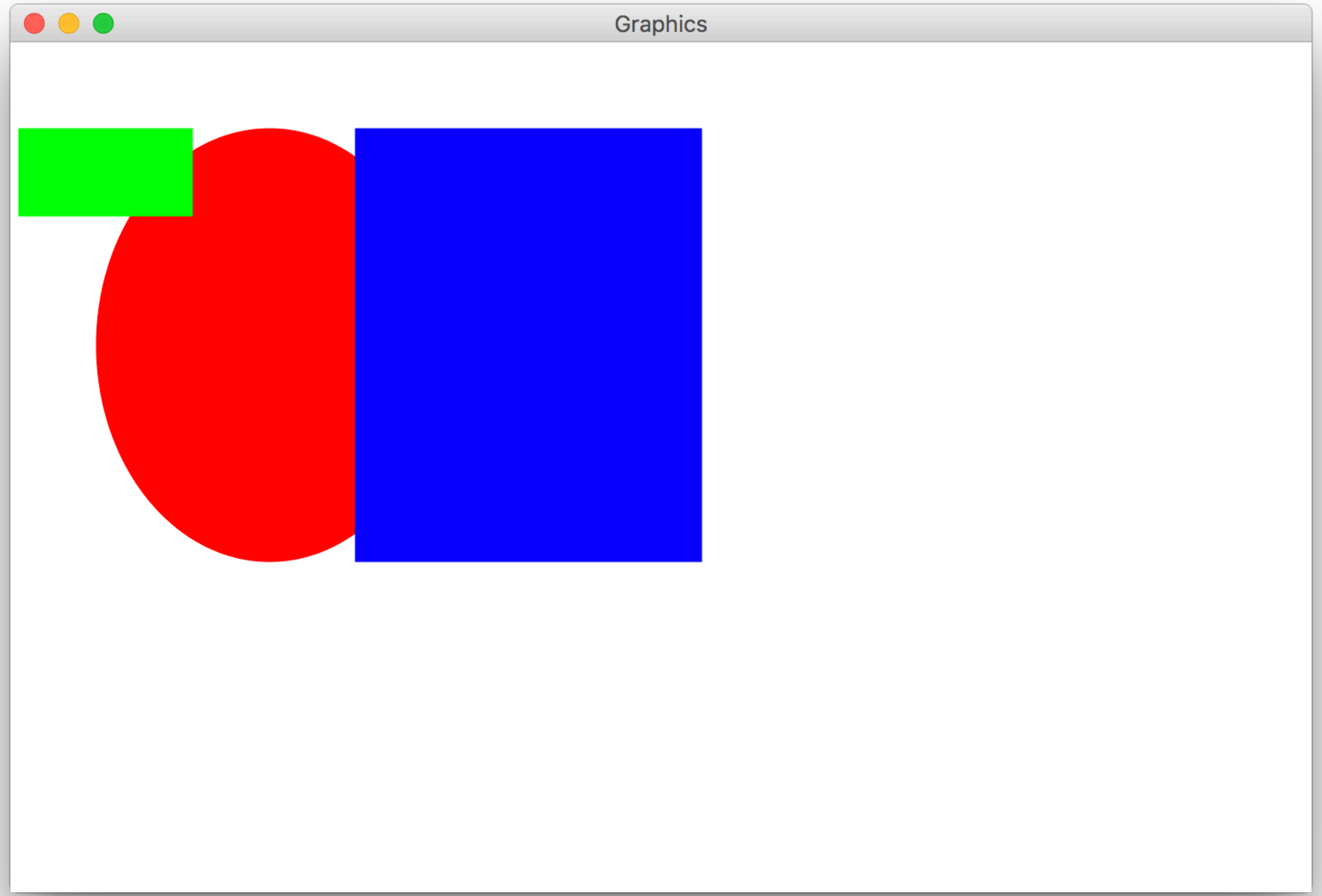
```
// Create a 200x250 GRect at (50, 50)
GRect rect = new GRect(50, 50, 200, 250);

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

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



The Graphics Canvas



Collage Model

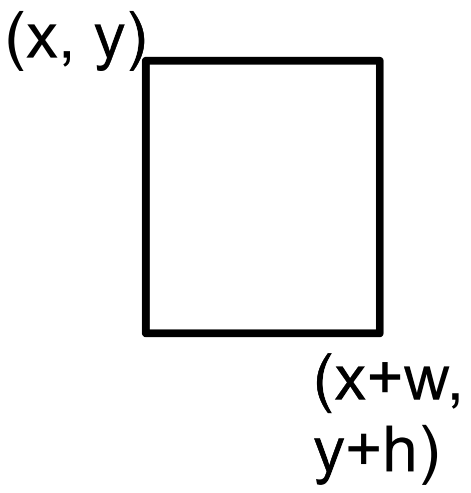


Plan For Today

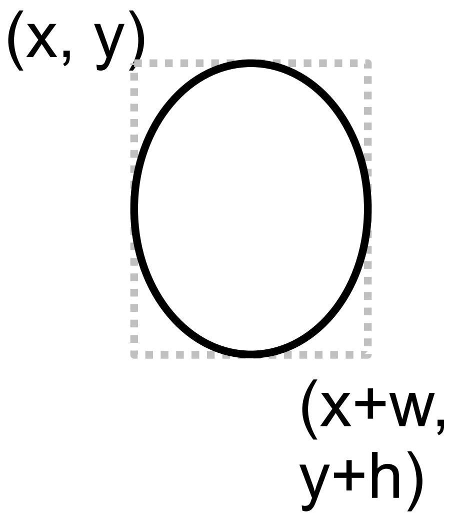
- Day 3 Overview
- Recap: Introduction to Java
- GraphicsProgram
- **Graphical Objects**
- Practice: Car

Graphical Objects

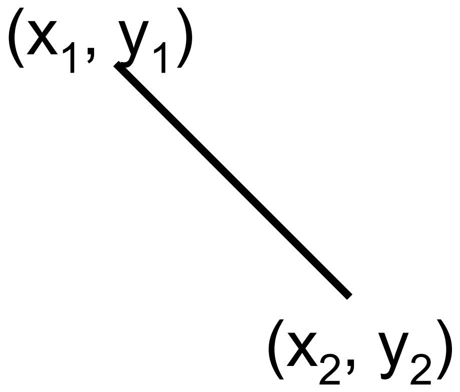
`GRect`



`GOval`



`GLine`

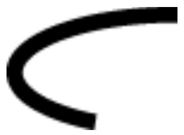


`GLabel`

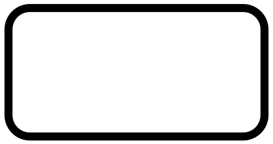
Hello there!

`GImage`

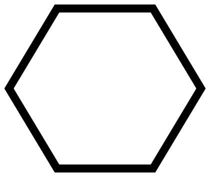
`GArc`



`GRoundRect`

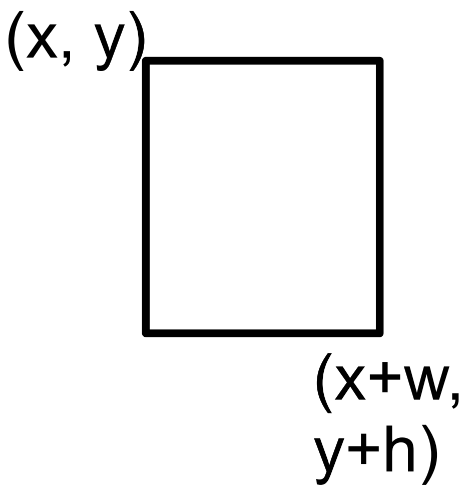


`GPolygon`

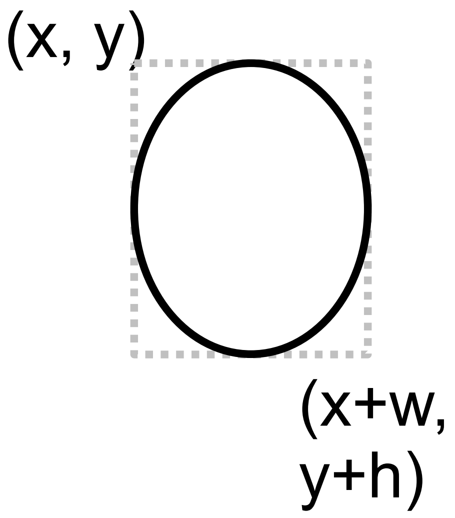


Graphical Objects

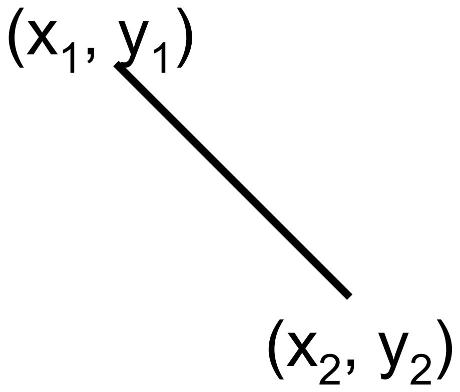
`GRect`



`GOval`



`GLine`



`GLabel`

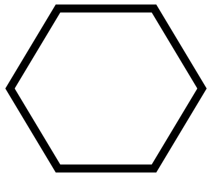
Hello there!

`GImage`

`GArc`

`GRoundRect`

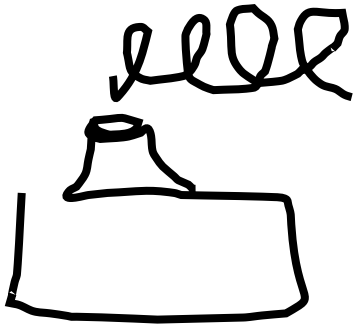
`GPolygon`



Graphical Objects

$$\begin{tikzcd}
 & \texttt{GObject} & & \\
\texttt{GLabel} \arrow[ru] & \texttt{GRect} \arrow[u] & \texttt{GOval} \arrow[lu] & \textit{others...} \arrow[llu]
\end{tikzcd}$$

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



Primitives vs. Objects

Primitive Variable Types	Object Variable Types
`int`	`GRect`
`double`	`GOval`
`char`	`GLine`
`boolean`	`GLabel`
	`...`

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:

$$\underbrace{\texttt{object}}_{\text{Receiver}}\texttt{.}\underbrace{\texttt{method(parameters);}}_{\text{Message}}$$

Methods on Graphics Objects

We manipulate graphics objects by calling methods on them:

$$\underbrace{\texttt{object}}_{\text{Who?}}\texttt{.}\underbrace{\texttt{method}}_{\text{What?}}\texttt{(}\underbrace{\texttt{parameters}}_{\text{What specifically?}}\texttt{);}$$

Example:

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

GObject Methods

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

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

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));
```

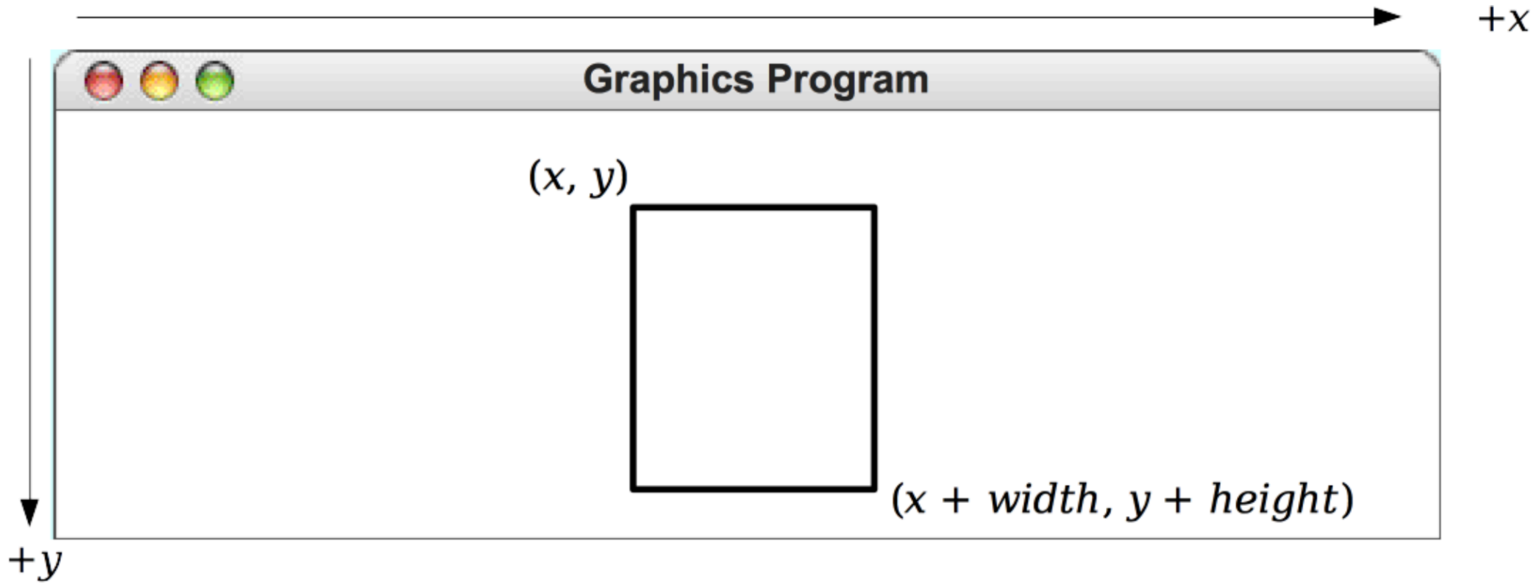
GRect

`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`*`);`

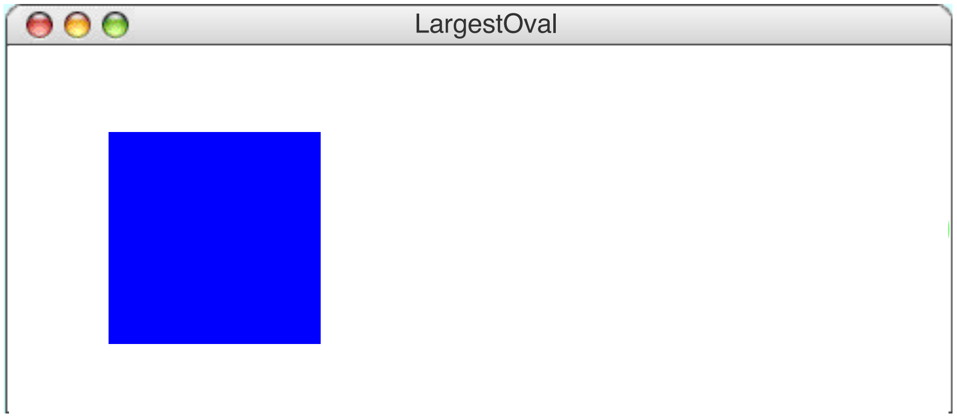
- Same as above, but defaults to (x, y) = (0, 0)



GRect

As an example, the following `run` method displays a rectangle

```
public void run() {
   Grect rect = new GRect(200, 200);
   rect.setFilled(true);
   rect.setColor(Color.BLUE);
   add(rect, 50, 50);
}
```



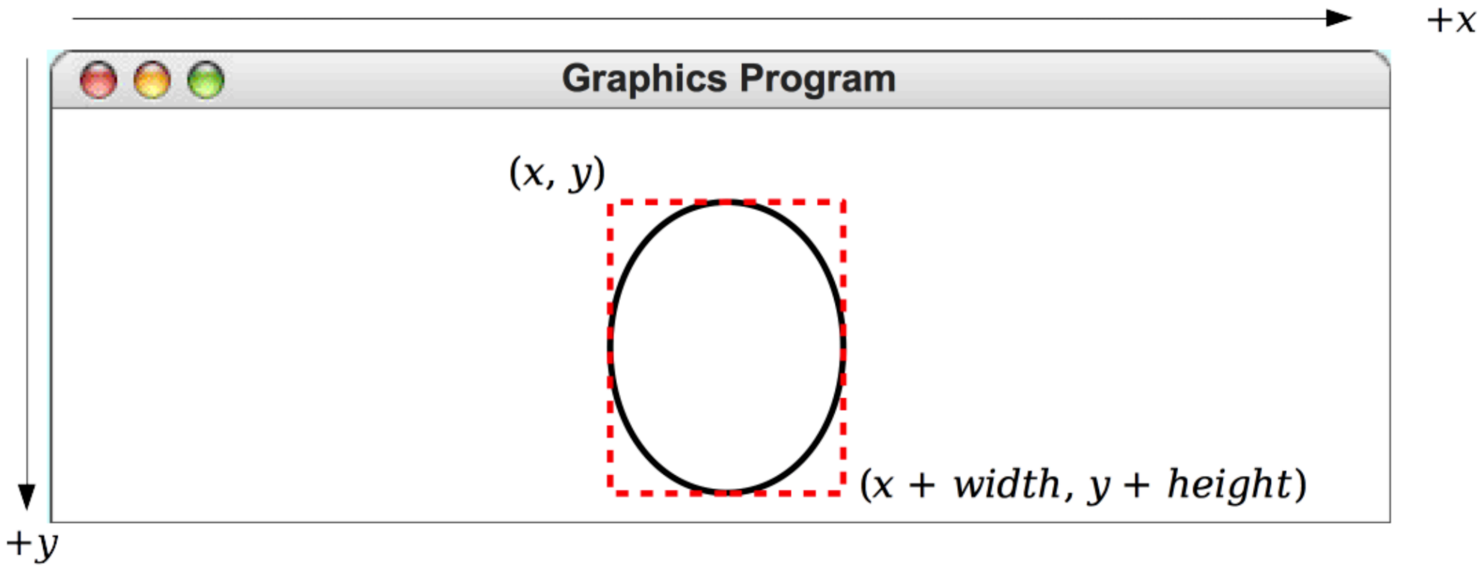
GOval

`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`*`);`

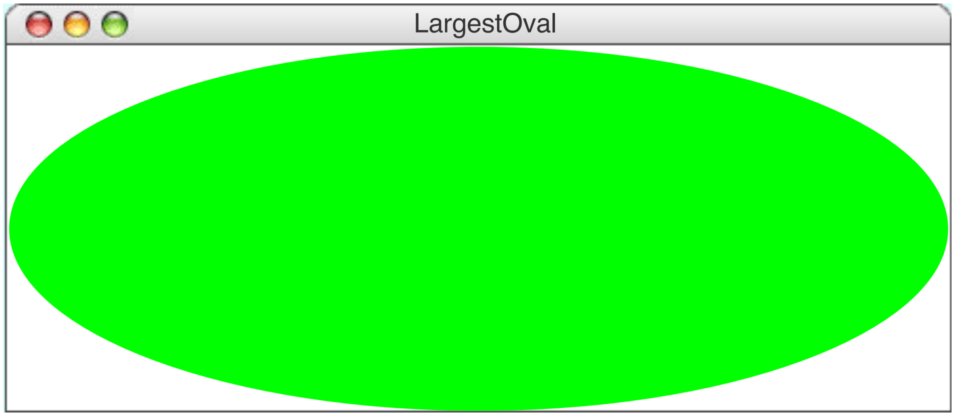
- Same as above, but defaults to (x, y) = (0, 0)



GOval

As an example, the following `run` method creates the largest oval that fits within the canvas:

```
public void run() {
   GOval oval = new GOval(getWidth(), getHeight());
   oval.setFilled(true);
   oval.setColor(Color.GREEN);
   add(oval, 0, 0);
}
```



GRect and GOval

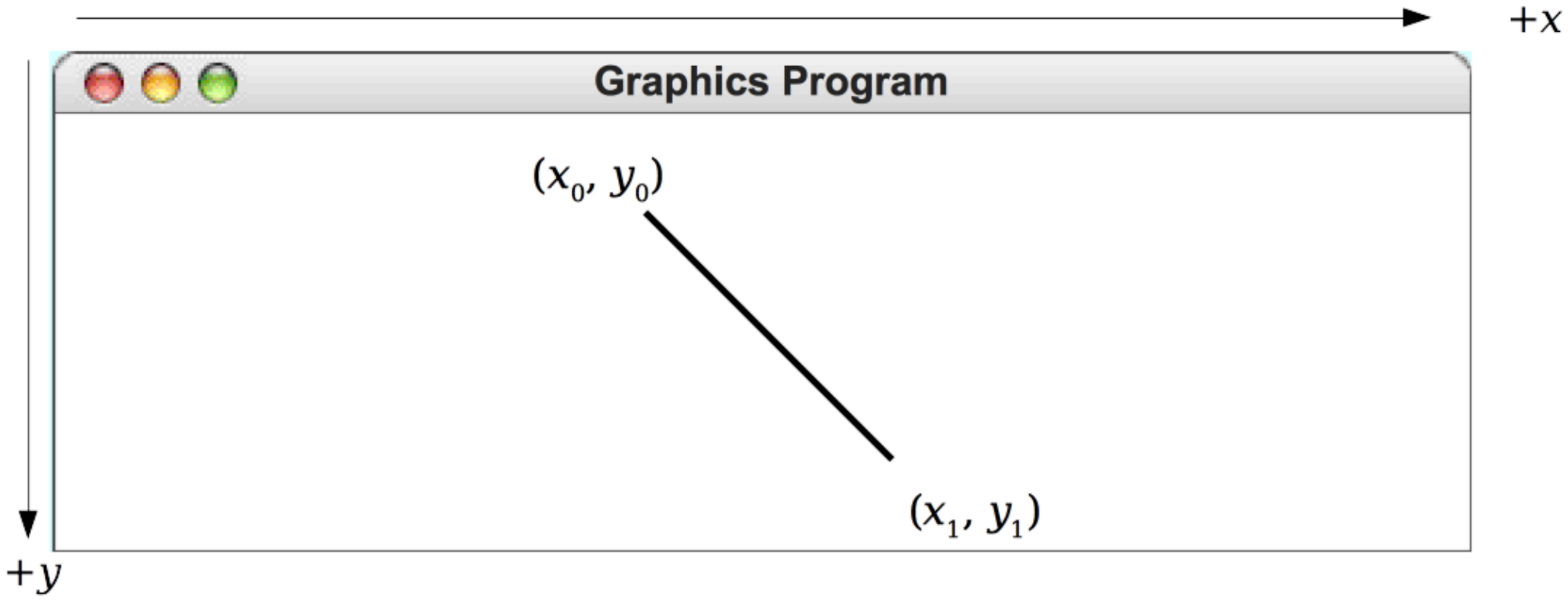
Methods shared by the **`GRect`** and **`GOval`** classes

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

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



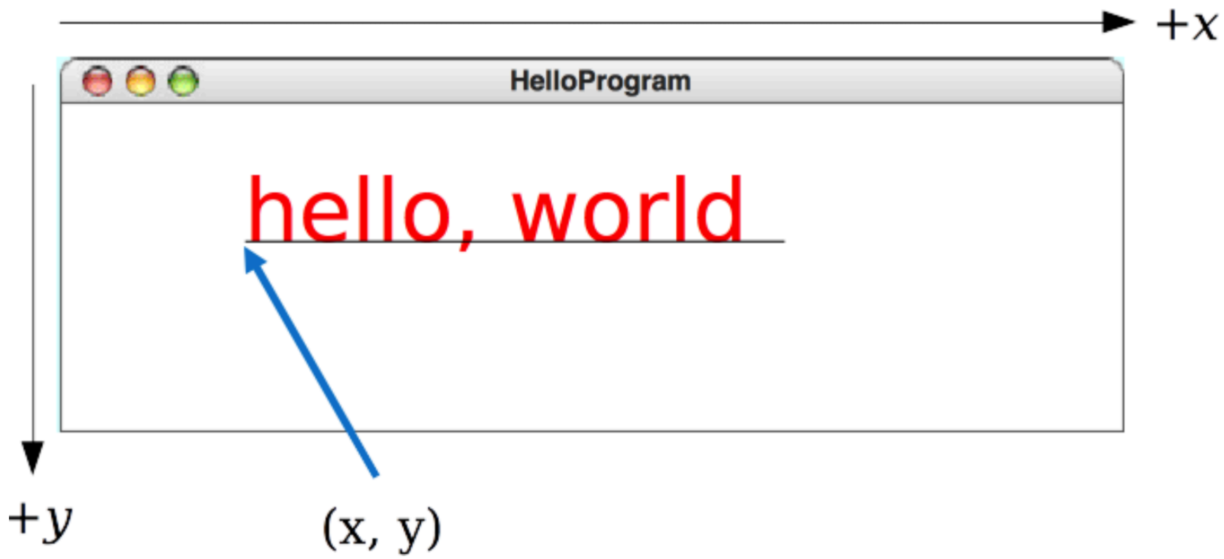
GLabel

`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`.`**`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*`"`

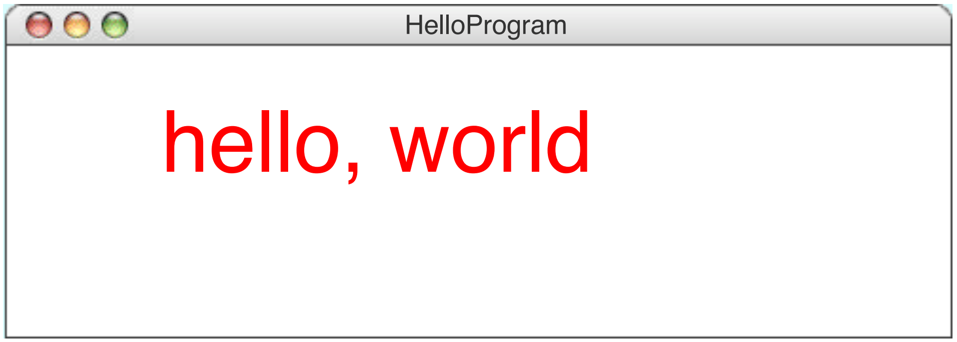
family is the name of a font family
style is either **`PLAIN`**, **`BOLD`**, **`ITALIC`**, or **`BOLDITALIC`**
size is an integer indicating the point size

Graphic courtesy of Eric Roberts

GLabel

A variable that represents text.

```
public class HelloProgram extends GraphicsProgram {
   public void run() {
      GLabel label = new GLabel("hello, world", 100, 75);
      label.setFont("SansSerif-36");
      label.setColor(Color.RED);
      add(label);
   }
}
```



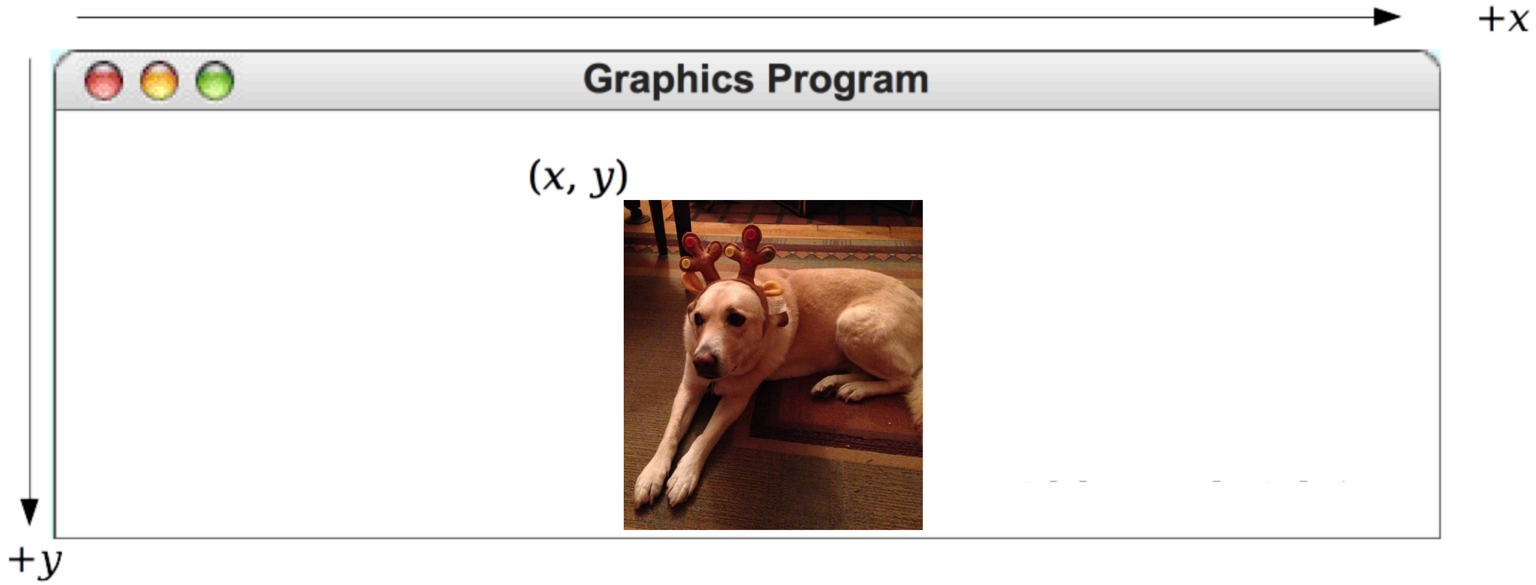
GImage

`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

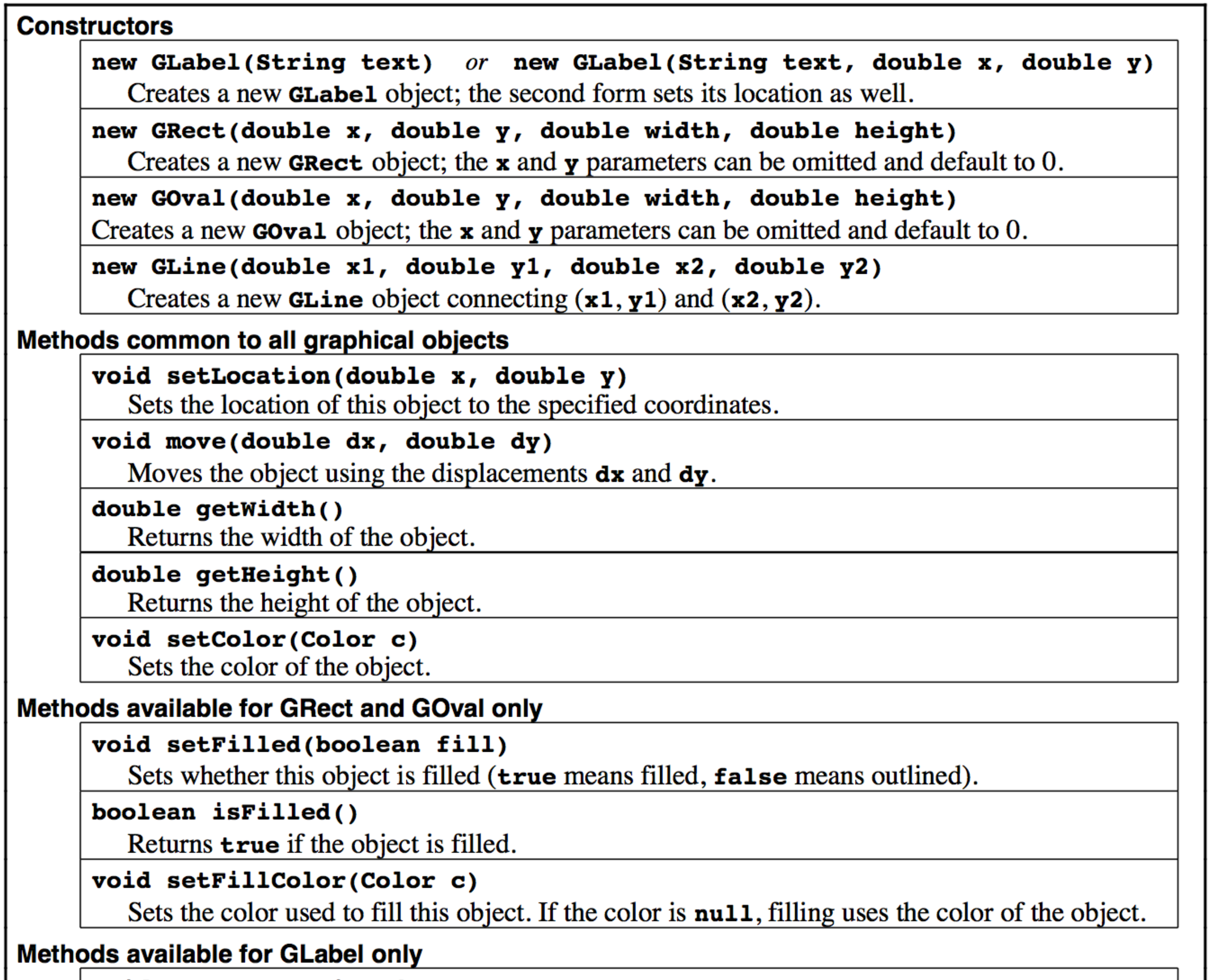
- `GraphicsProgram` contains these useful methods:

Method	Description

`add(`***gobj***`);`
`add(`***gobj***`, `***x***`, `***y***`);`
	adds a graphical object to the window

`getElementAt(`***x***`, `***y***`)`	return the object at the given (x,y) position(s)
`getWidth(), getHeight()`	return dimensions of window
`remove(`***gobj***`);`	removes a graphical object from the window
`removeAll();`	remove all graphical objects from window
`setBackground(`***color***`);`	set window's background color
`waitForClick()`	Suspends the program until the user clicks the mouse
`pause(`*ms*`)`	Pauses the program for *ms* milliseconds

Reference Sheet



Constructors

`new GLabel(String text)` *or* **`new GLabel(String text, double x, double y)`**
Creates a new **`GLabel`** object; the second form sets its location as well.

`new GRect(double x, double y, double width, double height)`
Creates a new **`GRect`** object; the **`x`** and **`y`** parameters can be omitted and default to 0.

`new GOval(double x, double y, double width, double height)`
Creates a new **`GOval`** object; the **`x`** and **`y`** parameters can be omitted and default to 0.

`new GLine(double x1, double y1, double x2, double y2)`
Creates a new **`GLine`** object connecting (**`x1`**, **`y1`**) and (**`x2`**, **`y2`**).

Methods common to all graphical objects

`void setLocation(double x, double y)`
Sets the location of this object to the specified coordinates.

`void move(double dx, double dy)`
Moves the object using the displacements **`dx`** and **`dy`**.

`double getWidth()`
Returns the width of the object.

`double getHeight()`
Returns the height of the object.

`void setColor(Color c)`
Sets the color of the object.

Methods available for GRect and GOval only

`void setFilled(boolean fill)`
Sets whether this object is filled (**`true`** means filled, **`false`** means outlined).

`boolean isFilled()`
Returns **`true`** if the object is filled.

`void setFillColor(Color c)`
Sets the color used to fill this object. If the color is **`null`**, filling uses the color of the object.

Methods available for GLabel only

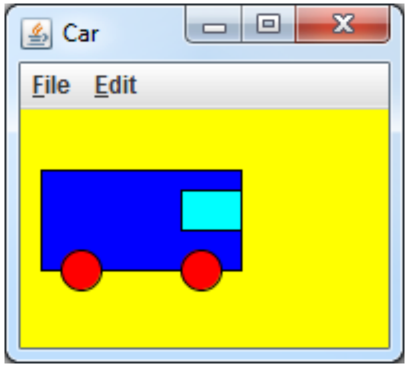
Plan For Today

- Day 3 Overview
- Recap: Introduction to Java
- GraphicsProgram
- Graphical Objects
- **Practice: Car**

Practice: Car

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

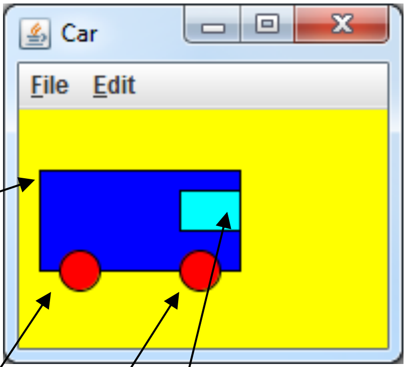
- Red wheels at (20, 70) and (80, 70), size 20x20
- Cyan windshield at (80, 40), size 30x20
- Blue body at (10, 30), size 100x50
- yellow background



Programming Time

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);
    }
}
```



Plan For Today

- Day 3 Overview
- Recap: Introduction to Java
- GraphicsProgram
- Graphical Objects
- Practice: Car