Graphics in Java

Announcements

- Sections start today! Hooray!
  - Section assignments emailed out earlier today.
- Assignment 1 (Karel) due this Friday.
  - Stop by the LaIR with questions!
- Email assignment due Sunday.
  - Looking forward to meeting you!

Testing Karel the Robot

LEAN IN
CHAPTER

CS @ Stanford

Have you ever been **intimidated** by CS?

Have you ever **struggled** to find a study group in CS?

Do you want to help build a **supportive** community?

Come to our info session to find out more

January 15th 7-8pm

Gates 415

RSVP to the Facebook event at http://on.fb.me/1ADcsK4

Outline for Today

- **Graphics in Java**
  - Oooh! Shiny!
- **Combining Expressions and Graphics**
  - Calculating with the Coordinate System
- **The `if` Statement Revisited**
  - Now with Variables!
- **The `for` Loop Revisited**
  - Now with Graphical Goodies!

Recap from Last Time

Variables

- A ***variable*** is a location where a program can store information for later use.

- Each variable has three pieces of information associated with it:
  - ***Name***: What is the variable called?
  - ***Type***: What sorts of things can you store in the variable?
  - ***Value***: What value does the variable have at any particular moment in time?

Declaring Variables

- In Java, before you can use a variable, you need to ***declare*** it so that Java knows the name, type, and value.
- The syntax for declaring a variable is

type name = ***value***;

- For example:
  - `int numVotes = 137;`
  - `double pricePerPound = 0.93;`

Programming with Graphics

Working with Graphics

- We will manipulate graphics on-screen by creating ***graphics objects*** and manipulating their properties.
- To create a graphics object, we need to
  - declare a variable to hold that object, and
  - actually create the object using the `new` keyword.
- For example:

```
GLabel label = new GLabel("Hi!", 0, 0);
```

Sending Messages

- You can manipulate graphics objects by calling methods on those objects.
- To call a method on an object, use the syntax

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

- For example:

```
label.setFont("Comic Sans-32");
label.setColor(Color.ORANGE);
```

Sending Messages

- You can manipulate graphics objects by calling methods on those objects.
- To call a method on an object, use the syntax

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

- For example:

```
label.setFont("Comic Sans-32");
label.setColor(Color.ORANGE);
```

Sending Messages

- You can manipulate graphics objects by calling methods on those objects.
- To call a method on an object, use the syntax

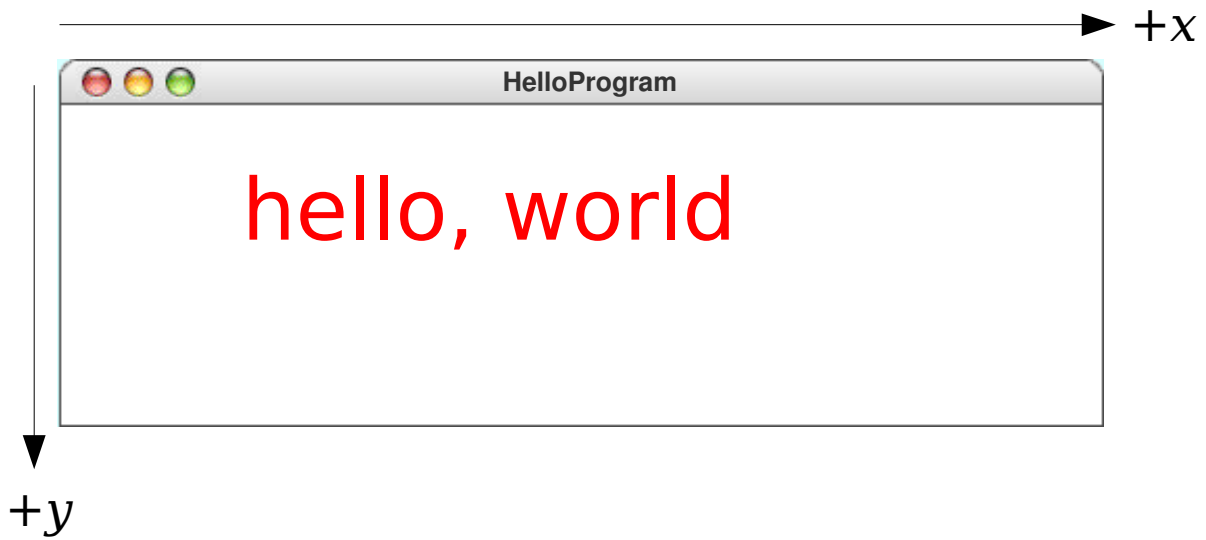
***object*.*method*(*parameters*)**

- For example:

```
label.setFont("Comic Sans-32");
label.setColor(Color.ORANGE);
```

Graphics Coordinates

- Graphics objects are positioned by specifying an x and y coordinate.
- x increases left-to-right, y increases top-to-bottom.
- x and y should be specified in pixels.
- For a `GLabel`, the x and y coordinates give the start of the ***baseline*** where the text is drawn.



Graphic courtesy of Eric Roberts

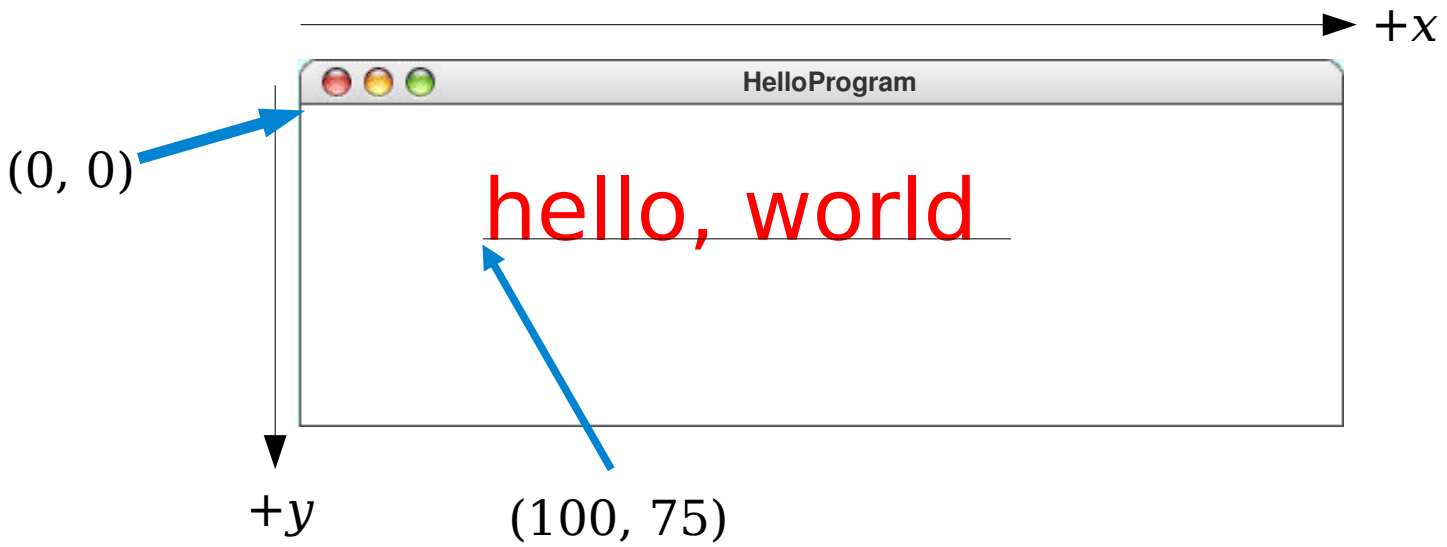
Graphics Coordinates

- Graphics objects are positioned by specifying an x and y coordinate.
- x increases left-to-right, y increases top-to-bottom.
- x and y should be specified in pixels.
- For a `GLabel`, the x and y coordinates give the start of the ***baseline*** where the text is drawn.

Graphic courtesy of Eric Roberts

Graphics Coordinates

- Graphics objects are positioned by specifying an *x* and *y* coordinate.
- *x* increases left-to-right, *y* increases top-to-bottom.
- *x* and *y* should be specified in pixels.
- For a `GLabel`, the *x* and *y* coordinates give the start of the ***baseline*** where the text is drawn.



Graphic courtesy of Eric Roberts

Drawing Geometrical Objects

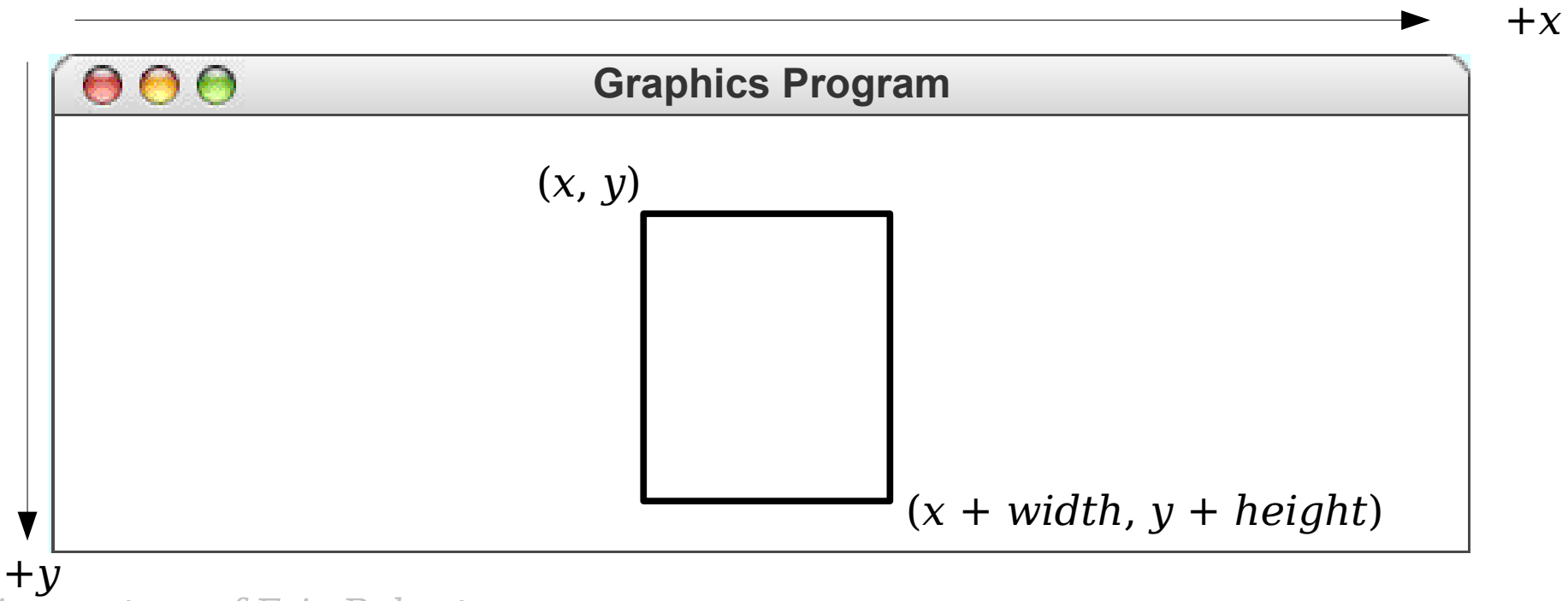
Graphic courtesy of Eric Roberts

Drawing Geometrical Objects

Creating graphics objects

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

Creates a rectangle whose upper left corner is at (*x*, *y*) of the specified size



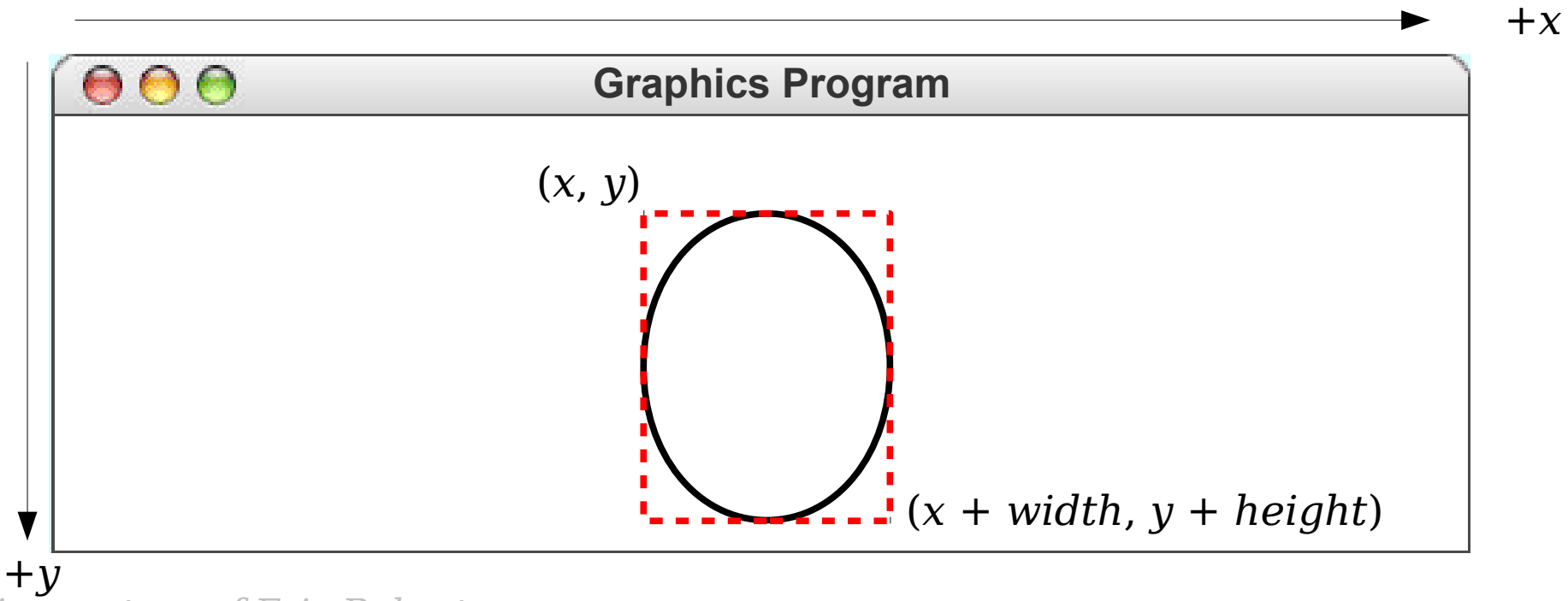
Graphic courtesy of Eric Roberts

Drawing Geometrical Objects

Creating graphics objects

`new GRect(` *x*, *y*, *width*, *height*`)`
Creates a rectangle whose upper left corner is at (*x*, *y*) of the specified size

`new GOval(` *x*, *y*, *width*, *height*`)`
Creates an oval that fits inside the rectangle with the same dimensions.



Graphic courtesy of Eric Roberts

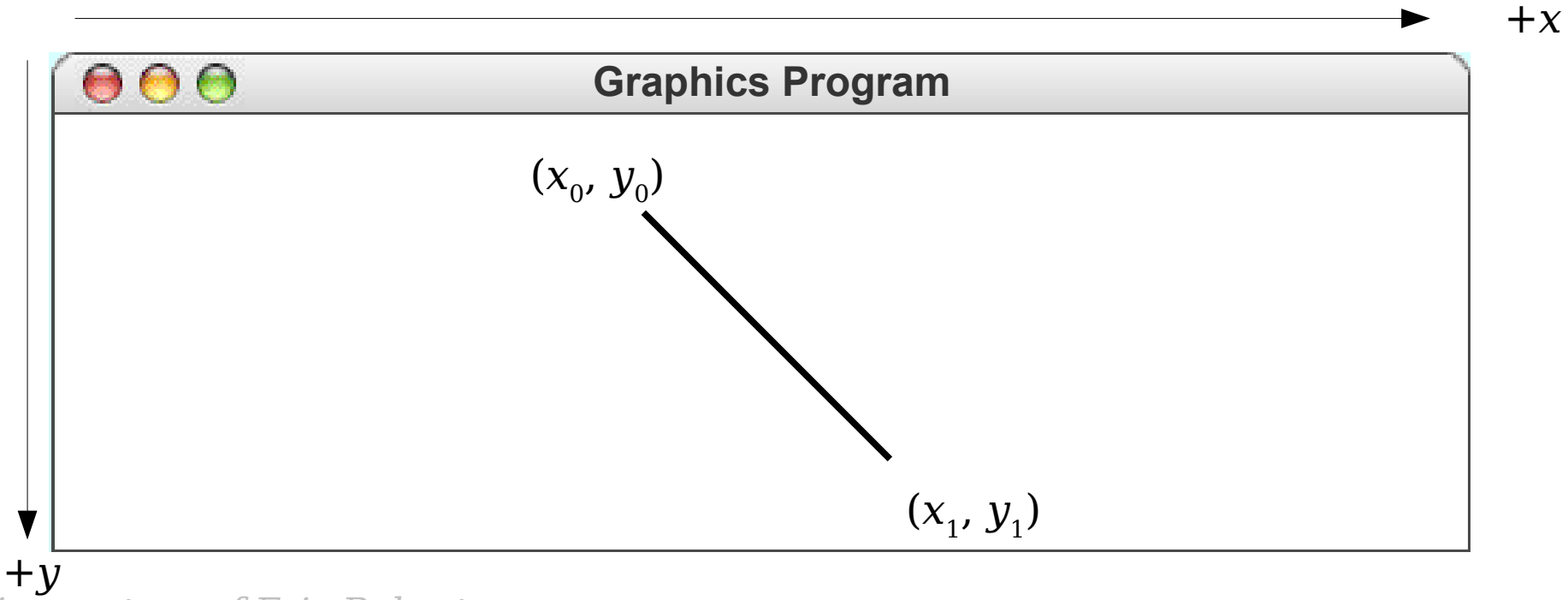
Drawing Geometrical Objects

Creating graphics objects

`new GRect(` *x*, *y*, *width*, *height*`)`
Creates a rectangle whose upper left corner is at (*x*, *y*) of the specified size

`new GOval(` *x*, *y*, *width*, *height*`)`
Creates an oval that fits inside the rectangle with the same dimensions.

`new GLine(` x_0, y_0, x_1, y_1`)`
Creates a line extending from (x_0, y_0) to (x_1, y_1).



Graphic courtesy of Eric Roberts

Drawing Geometrical Objects

Creating graphics objects

`new GRect(` *x*, *y*, *width*, *height*`)`
Creates a rectangle whose upper left corner is at (*x*, *y*) of the specified size

`new GOval(` *x*, *y*, *width*, *height*`)`
Creates an oval that fits inside the rectangle with the same dimensions.

`new GLine(` x_0, y_0, x_1, y_1`)`
Creates a line extending from (x_0, y_0) to (x_1, y_1).

Methods shared by `GRect` and `GOval`

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.

Graphic courtesy of Eric Roberts

The Collage Model

The Collage Model

Computing with Graphics

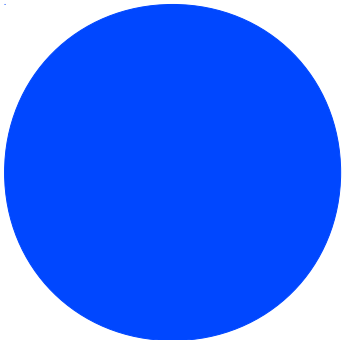
Size of the Graphics Window

Methods provided by **`GraphicsProgram`**

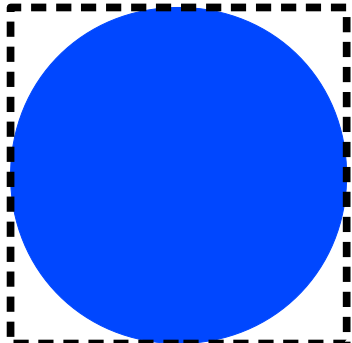
`getWidth()` Returns the width of the graphics window.
`getHeight()` Returns the height of the graphics window.

Like `println`, `readInt`, and `readDouble`, you don't need to prefix these methods with the ***object*****.** notation.

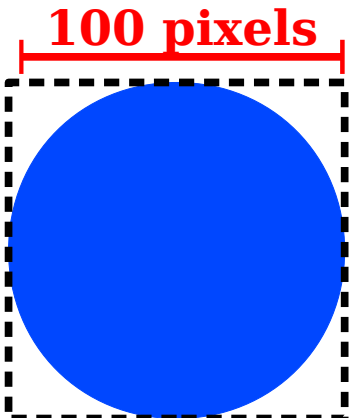
Making This Circle



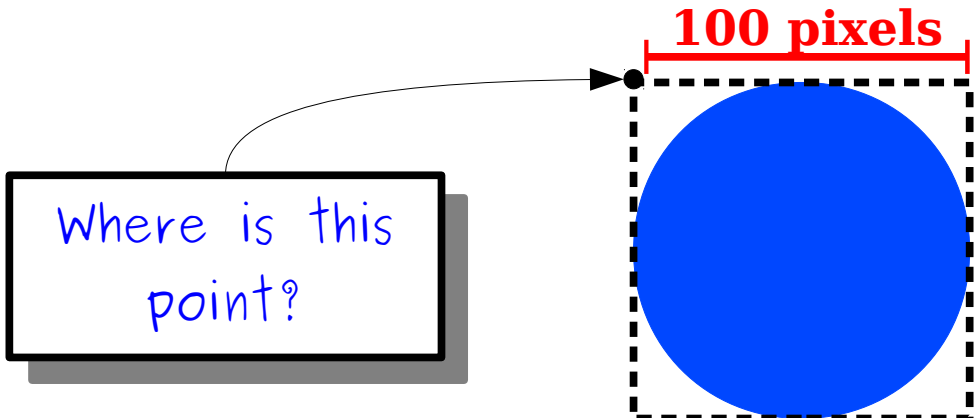
Making This Circle



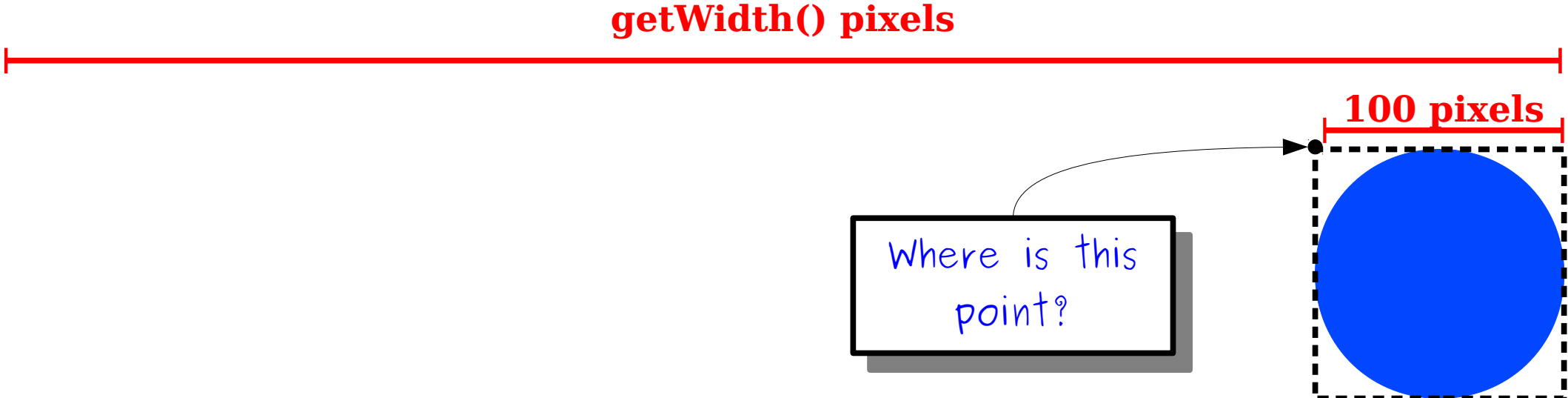
Making This Circle



Making This Circle

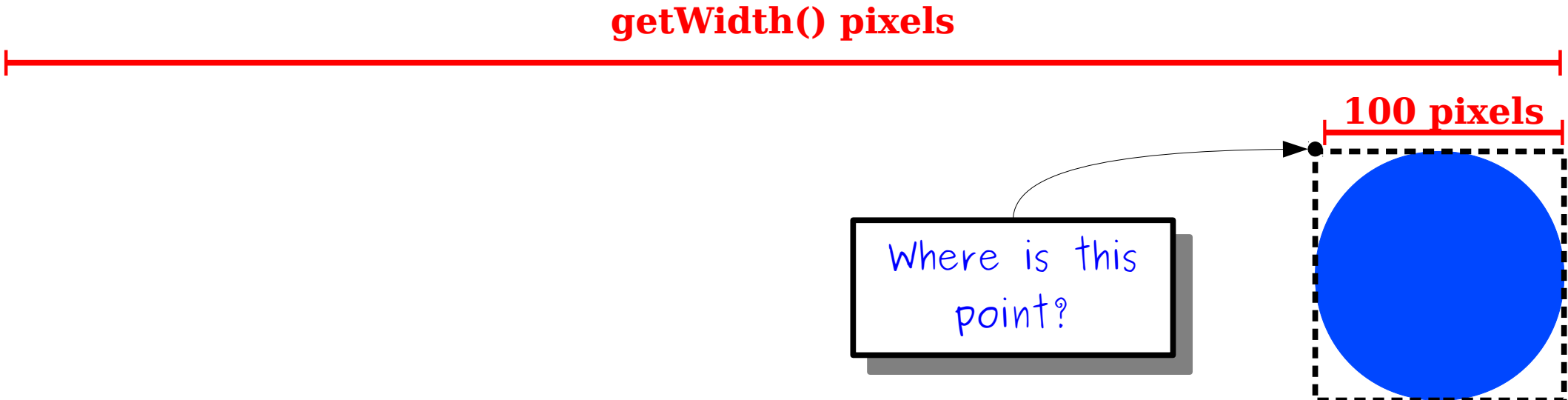


Making This Circle



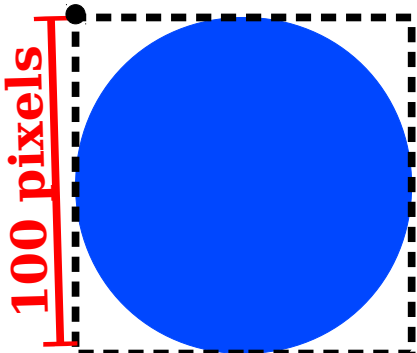
Making This Circle

```
double x = getWidth() - 100;
```



Making This Circle

```
double x = getWidth() - 100;
```



Making This Circle

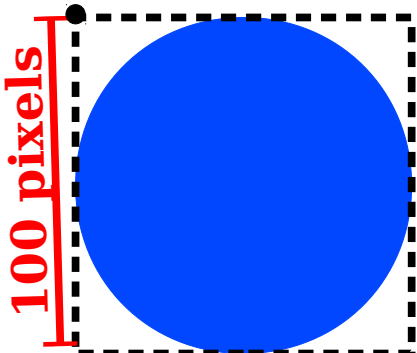
```
double x = getWidth() - 100;
```

Making This Circle

```
double x = getWidth() - 100;
double y = getHeight() - 100;
```

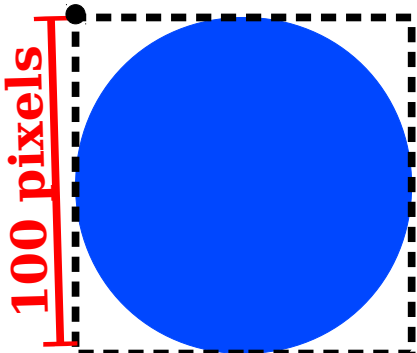
Making This Circle

```
double x = getWidth() - 100;
double y = getHeight() - 100;
```



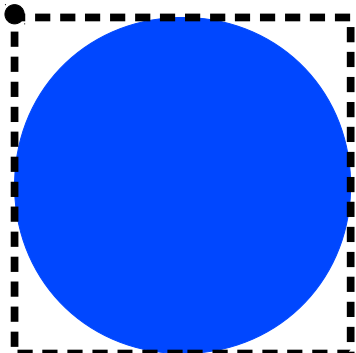
Making This Circle

```
double x = getWidth() - 100;
double y = getHeight() - 100;

GOval circle = new GOval(x, y, 100, 100);
```



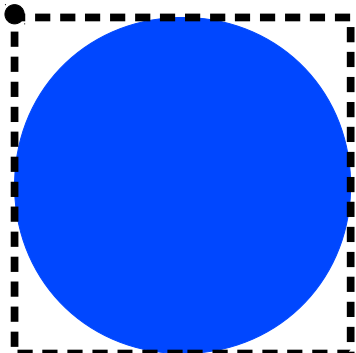
Making This Circle

```
double x = getWidth() - 100;
double y = getHeight() - 100;

GOval circle = new GOval(x, y, 100, 100);
```



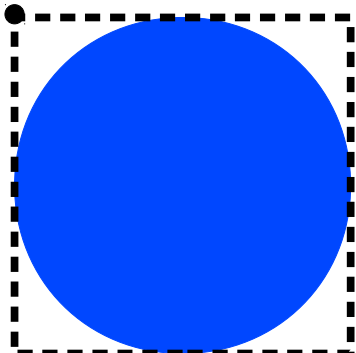
Making This Circle

```
double x = getWidth() - 100;
double y = getHeight() - 100;

GOval circle = new GOval(x, y, 100, 100);
circle.setFilled(true);
circle.setColor(Color.BLUE);
```



Making This Circle

```
double x = getWidth() - 100;
double y = getHeight() - 100;

GOval circle = new GOval(x, y, 100, 100);
circle.setFilled(true);
circle.setColor(Color.BLUE);
add(circle);
```



Magic Numbers

- A ***magic number*** is a number written in a piece of code whose meaning cannot easily be deduced from context.

```
double weight = 9.8 * (mass - 14.3);
```

- Magic numbers are considered poor style for a few reasons:
  - They decrease readability.
  - They complicate maintenance.
- Good heuristic: numbers other than 0, 1, and 2 are *usually* magic numbers.

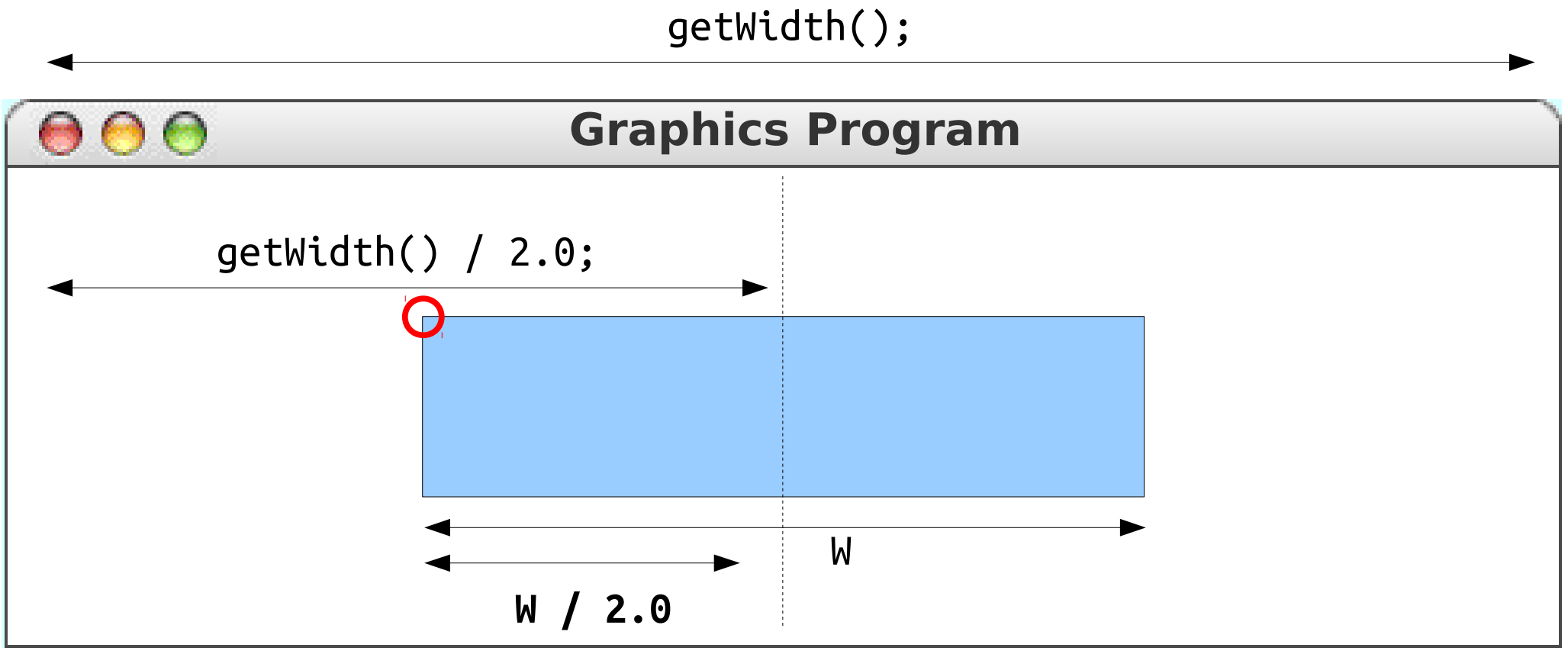
Constants

- A ***constant*** is a name for a value that never changes.
- Syntax (defined outside of any method):

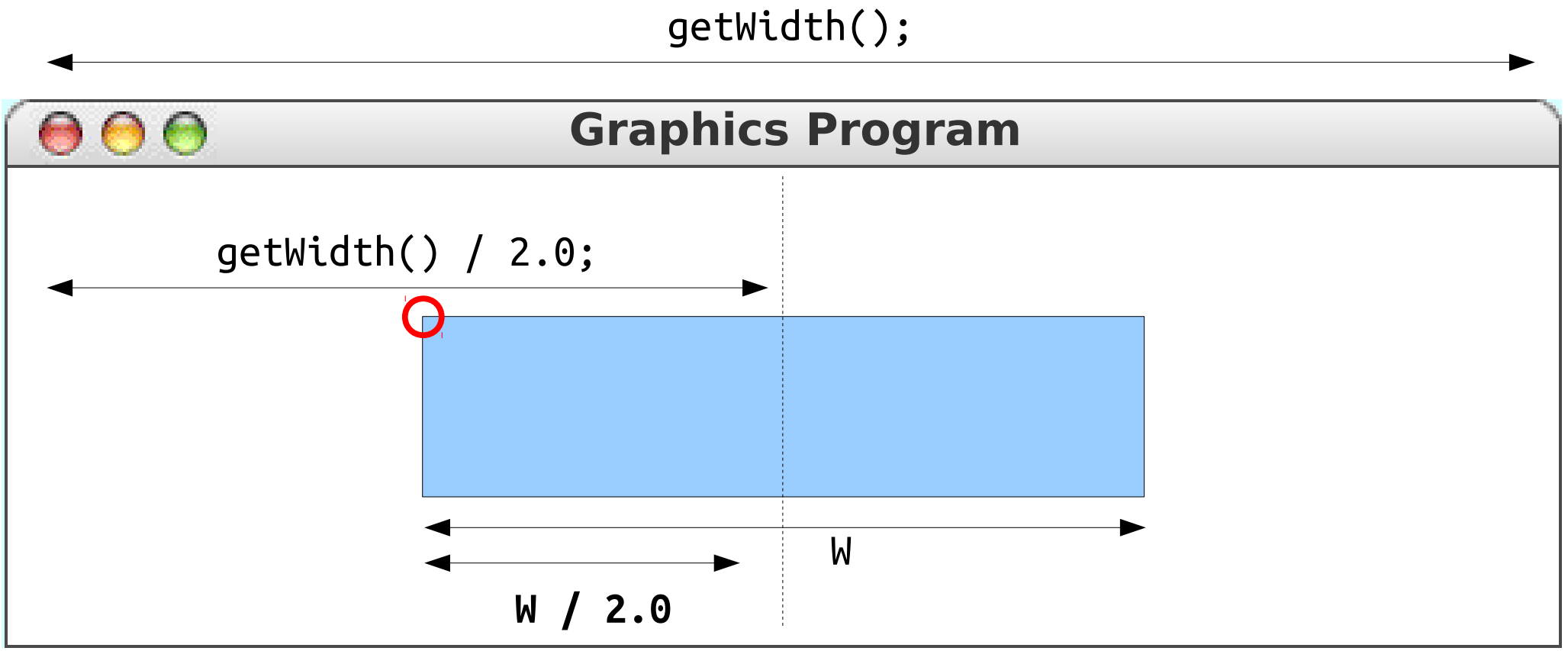
```
private static final type name = value;
```

- By convention, constants are named in `UPPER_CASE_WITH_UNDERSCORES` to differentiate them from variables.
- Constants can significantly improve code readability. They also improve code maintainability.

Centering an Object



Centering an Object




```
double x = (getWidth() / 2.0) - (W / 2.0);
```

- *or* -

```
double x = (getWidth() - W) / 2.0;
```