Methods

Friday Four Square Today!

Gates, 4:15PM

Drawing Geometrical Objects

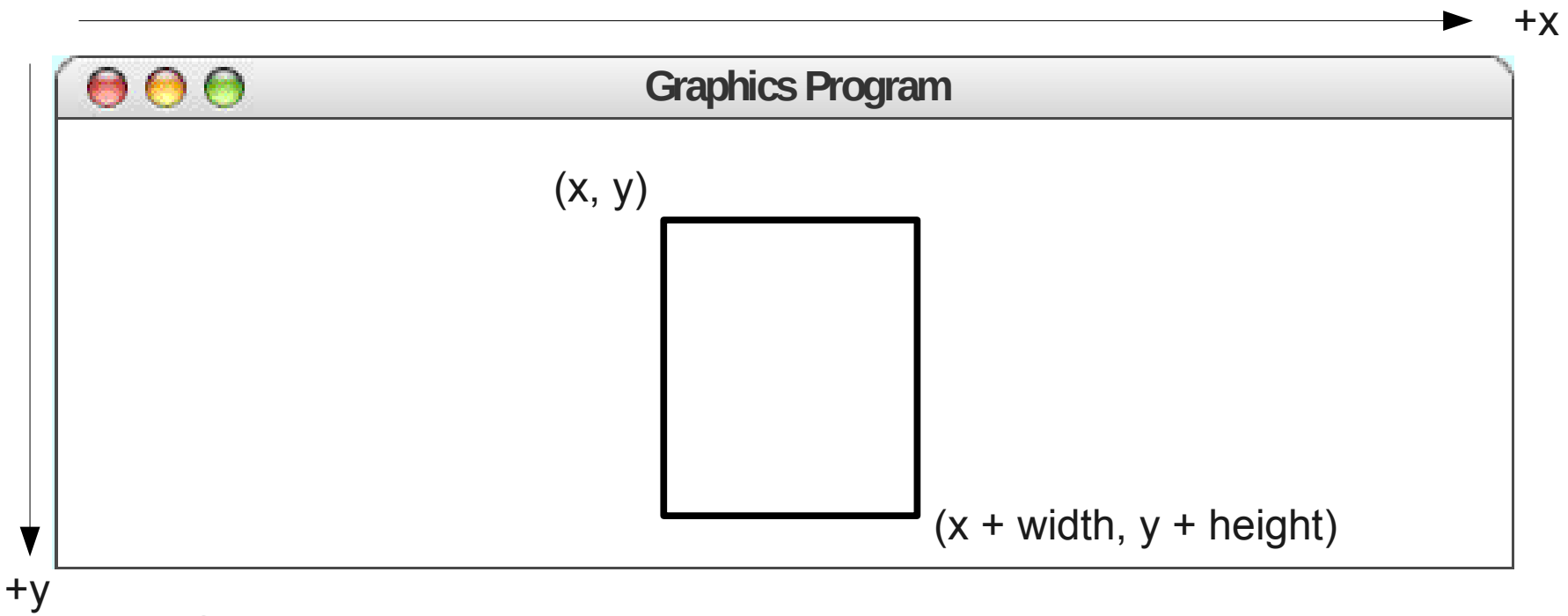
Graphic courtesy of Eric Roberts

Drawing Geometrical Objects

Constructors

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

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



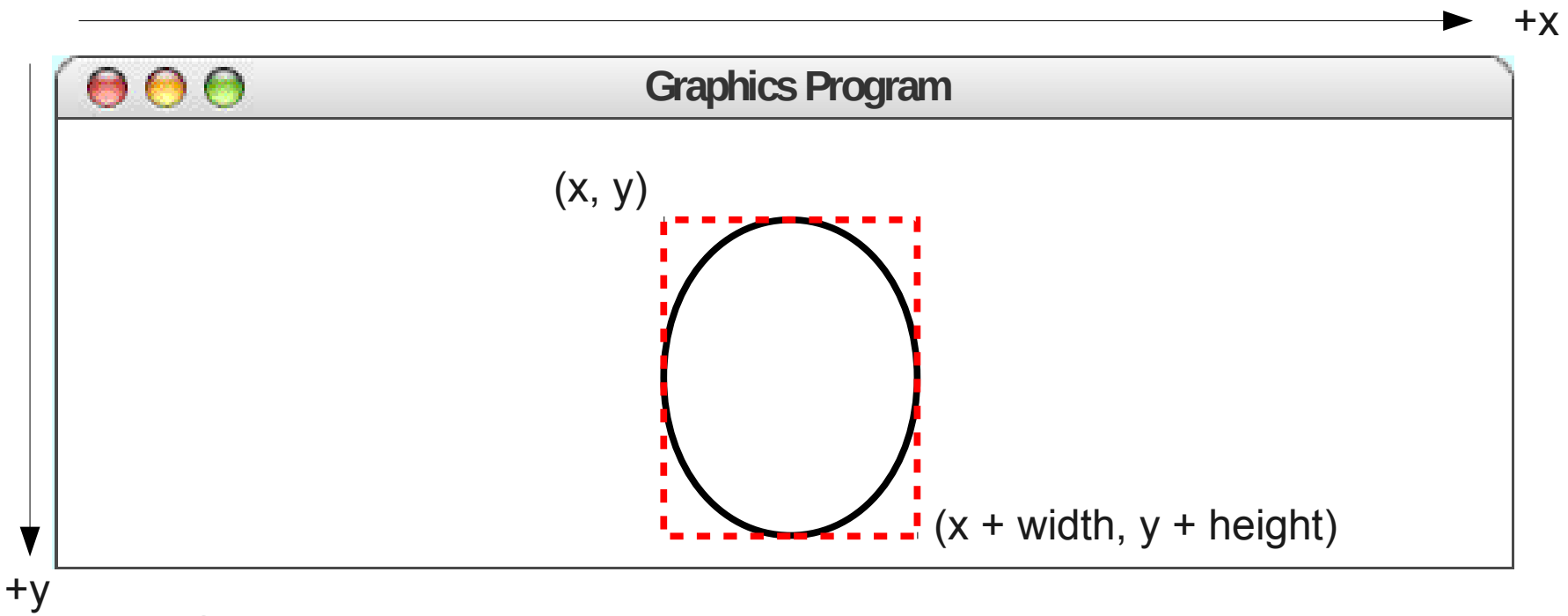
Graphic courtesy of Eric Roberts

Drawing Geometrical Objects

Constructors

`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

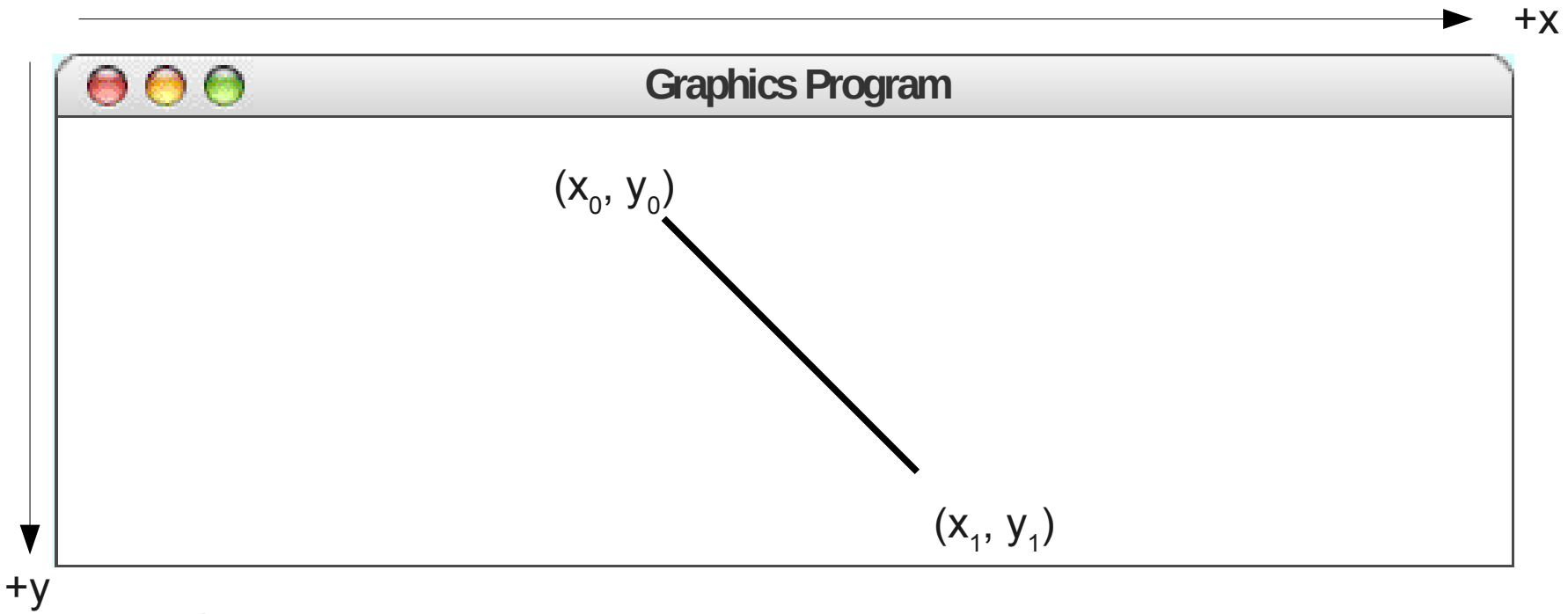
Drawing Geometrical Objects

Constructors

`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

Constructors

`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 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.

Graphic courtesy of Eric Roberts

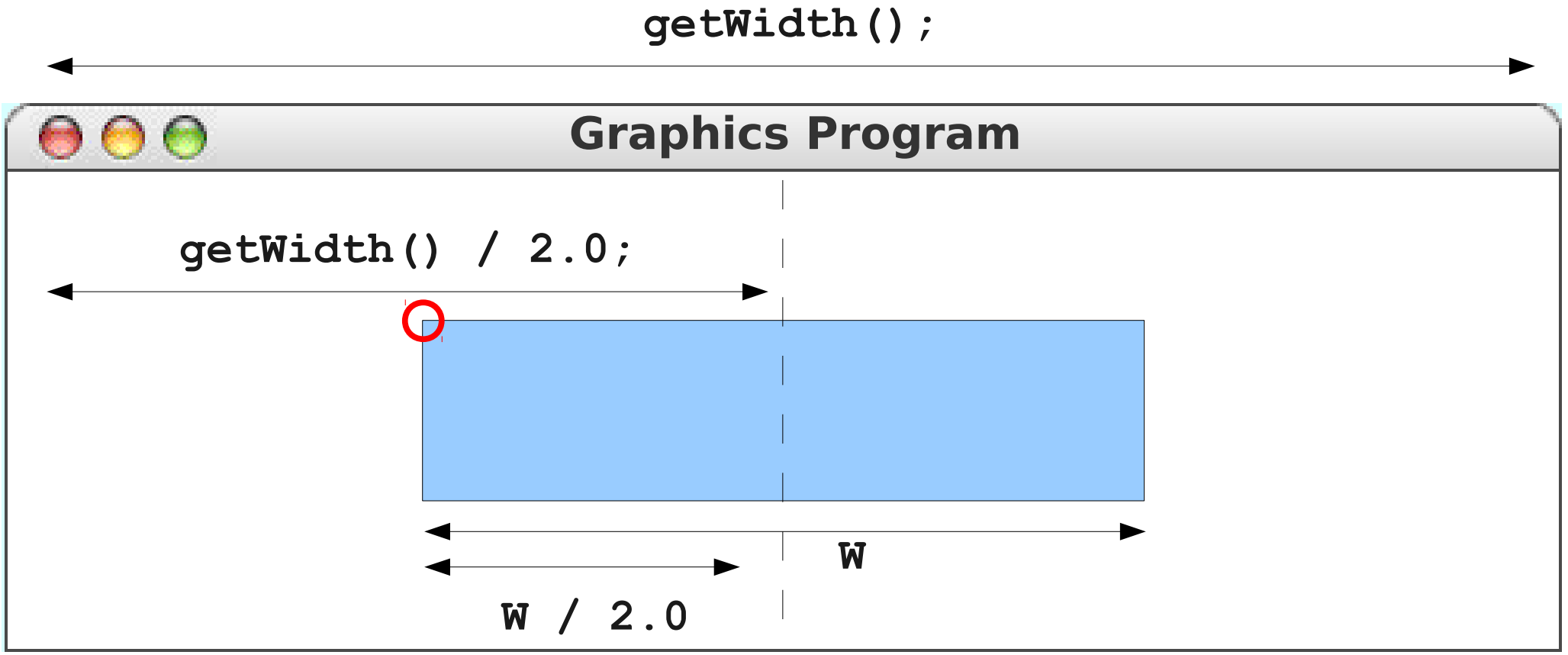
Size of the Graphics Window

Methods provided by **`GraphicsProgram`** class

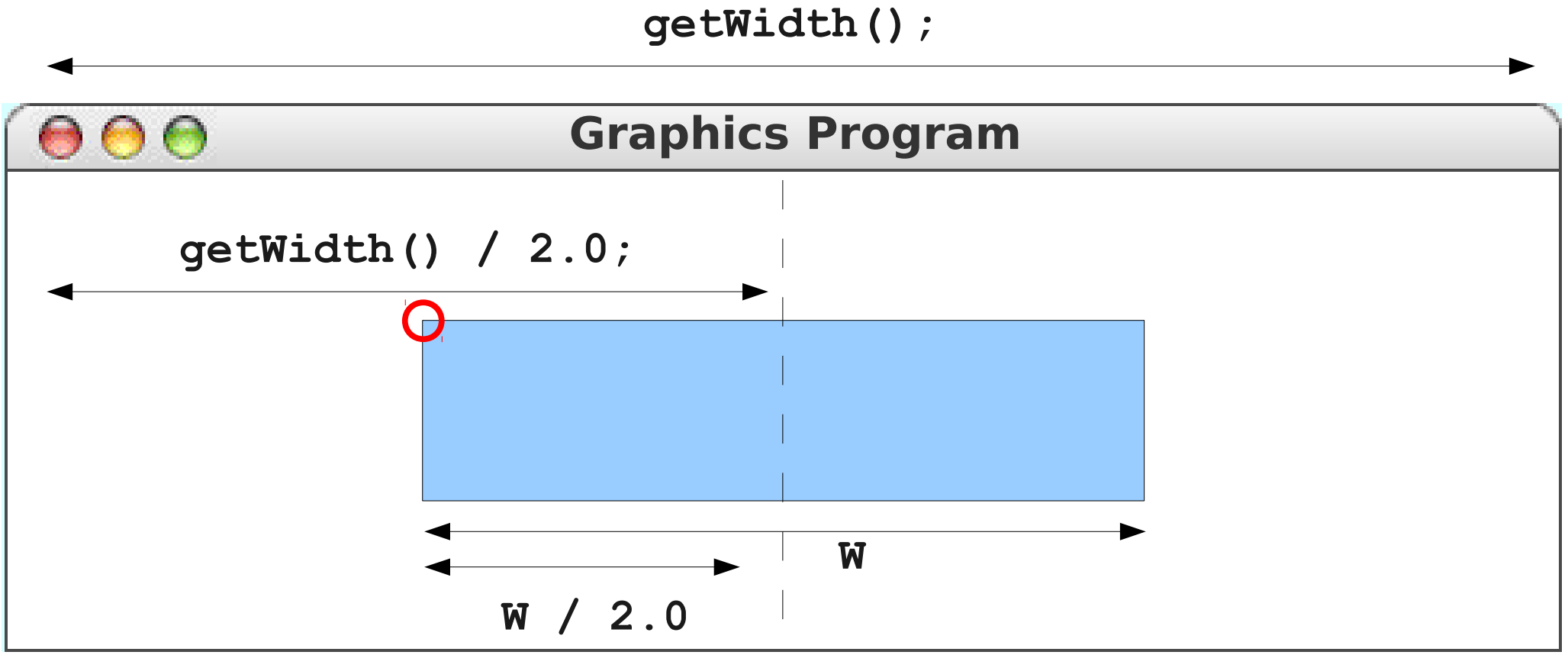
`getWidth()` Returns the width of the graphics window.
`getHeight()` Returns the height of the graphics window.

Note: receiver of these calls is the **`GraphicsProgram`** itself, so we don't need to specify a separate object as receiver.

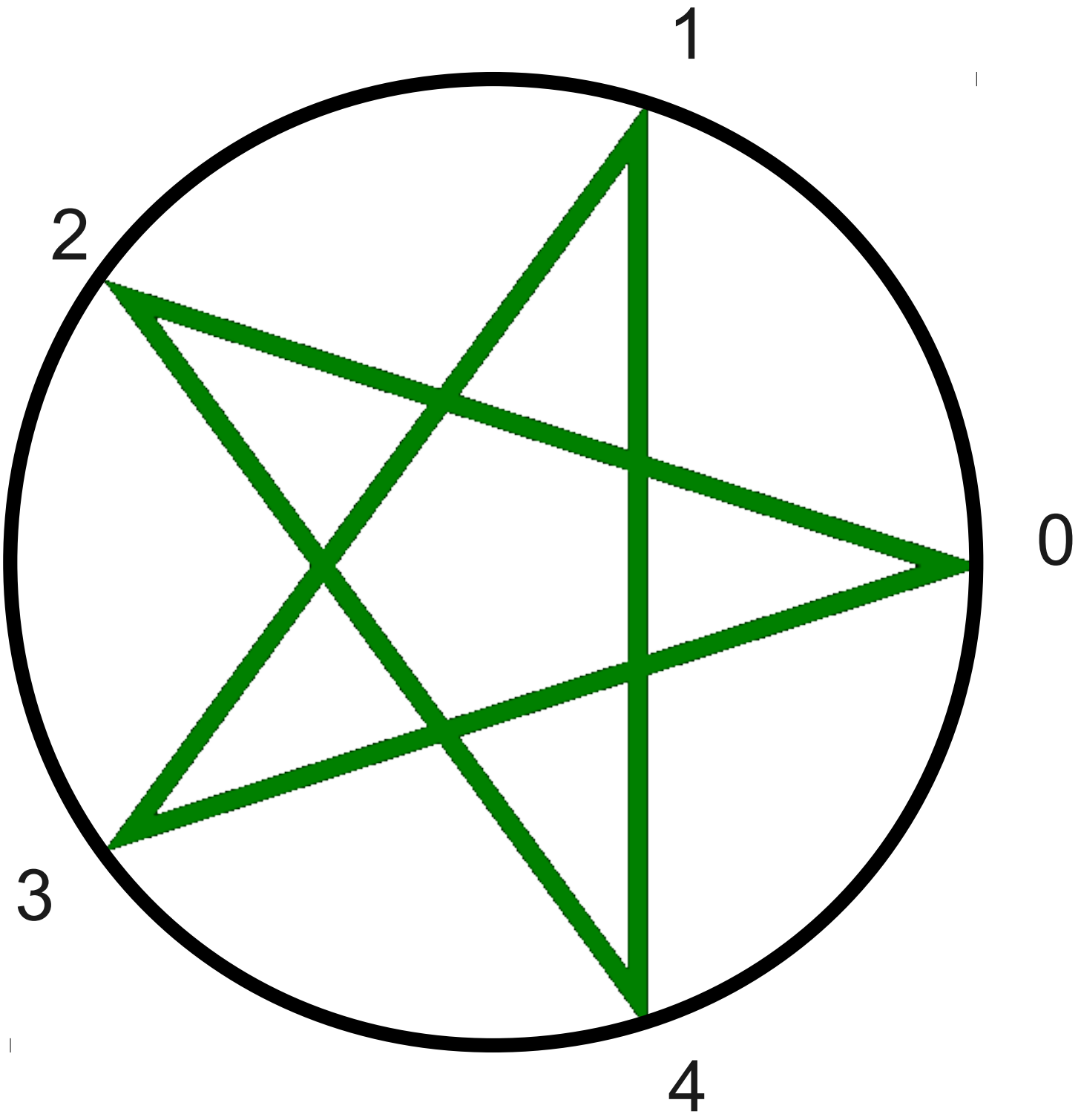
Centering an Object

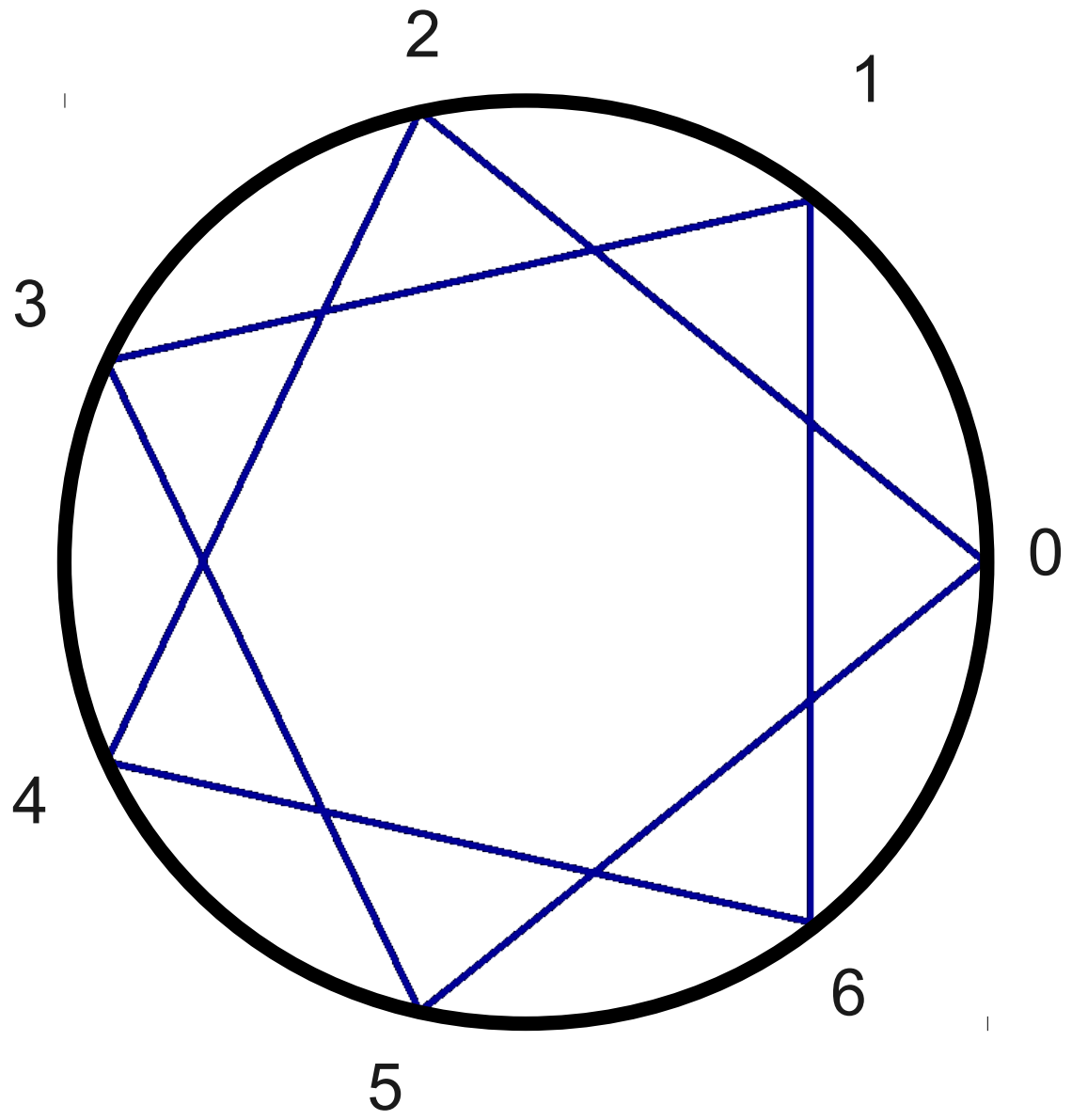


Centering an Object

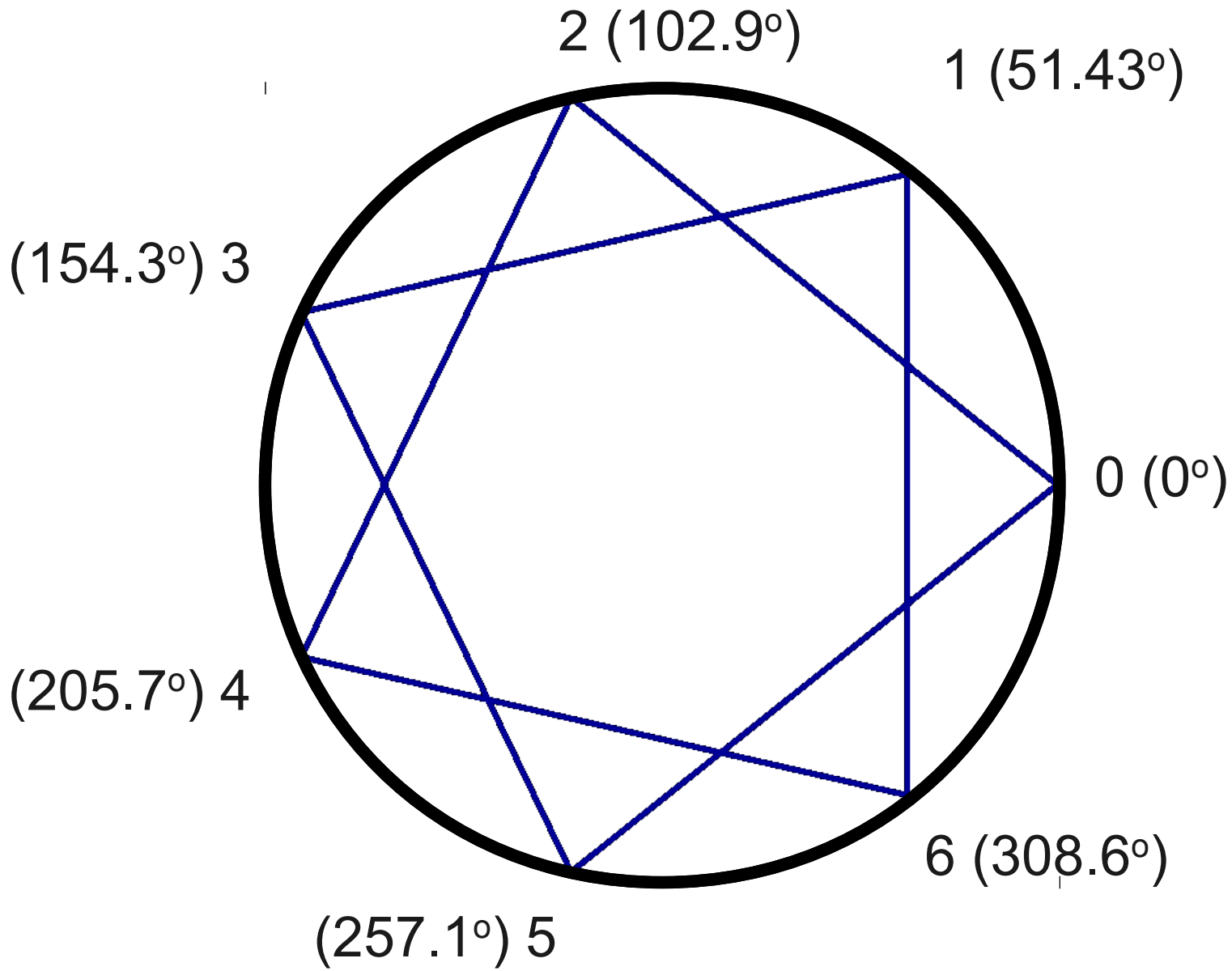



```
x = (getWidth() / 2.0) - (W / 2.0);
x = (getWidth() - W) / 2.0;
```


1
2
0
3
4

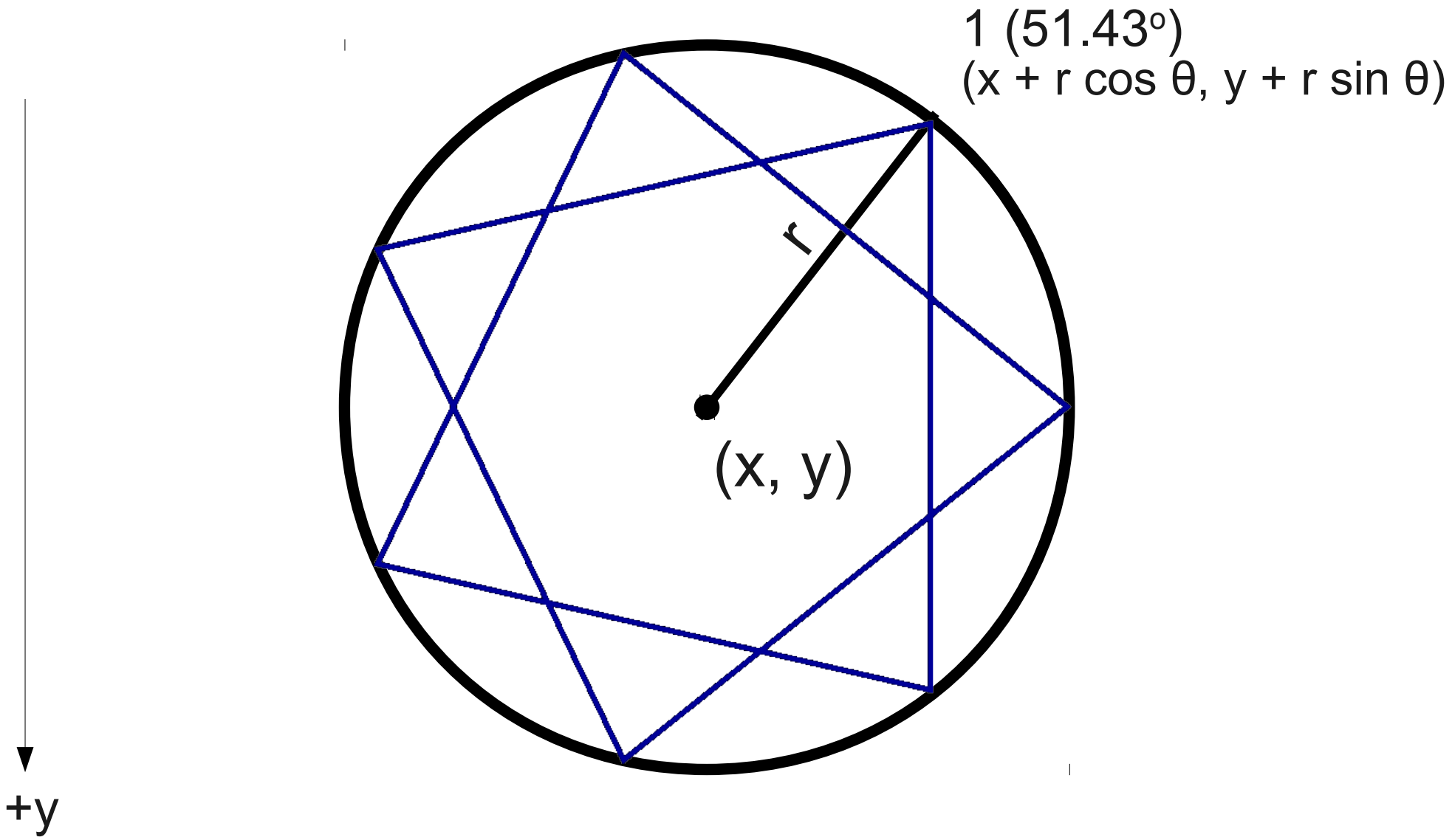


Each point k is connected to point k + 2, after wrapping around.



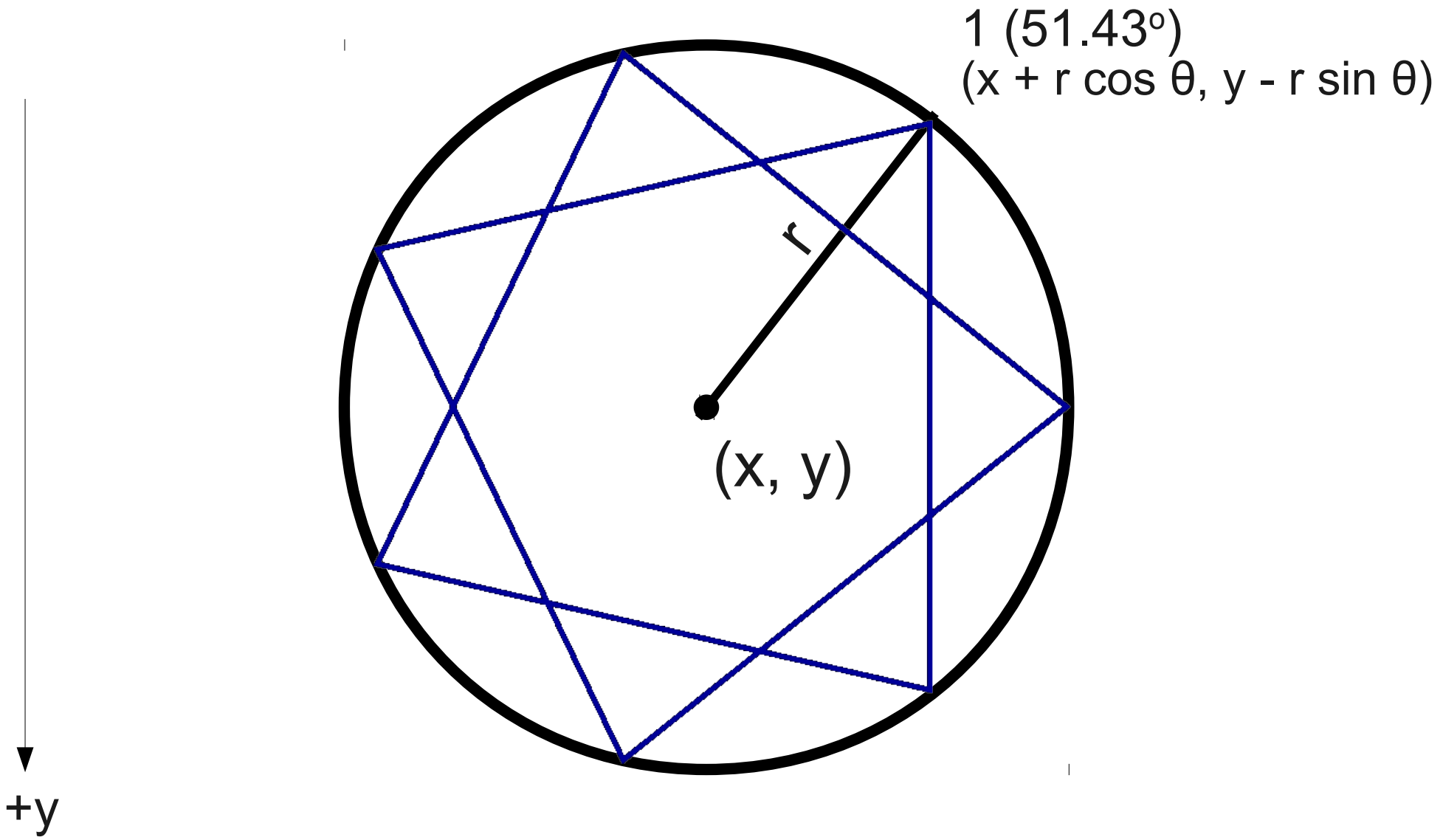
Each point k is connected to point k + 2, after wrapping around.

Point k is at $\frac{k}{numSides} \times 360^{\circ}$



Each point k is connected to point k + 2, after wrapping around.

Point k is at $\frac{k}{numSides} \times 360^{\circ}$



Each point k is connected to point k + 2, after wrapping around.

Point k is at $\frac{k}{numSides} \times 360^{\circ}$

Passing Parameters

- A method can accept **parameters** when it is called.
- Syntax:

```
private void name(parameters) {
    /* … method body … */
}
```

- The values of the parameters inside the method are set when the method is called.
- The values of the parameters can vary between calls.

For more on the geometry
and properties of stars:

Vi Hart on Stars:
http://youtu.be/CfJzrmS9UfY

Wikipedia on Stars:
http://en.wikipedia.org/wiki/Star_polygon

Factorials

- The number ***n* factorial**, denoted ***n*!**, is

$$1 \times 2 \times 3 \times \ldots \times (n - 1) \times n$$

- For example:
  - $3! = 1 \times 2 \times 3 = 6$.
  - $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$
  - $0! = 1$ (by definition)
- Factorials show up everywhere:
  - Taylor series.
  - Counting ways to shuffle a deck of cards.
  - Determining how quickly computers can sort values.

Returning Values

- A method may produce a value that can be read by its caller.
- To indicate that a method returns a value, specify the type returned in the method declaration:

```
private type name(parameters) {
    /* … method body … */
}
```

- A value can be returned with the **`return`** statement:

```
return value;
```

Subtleties of **`return`**

- If a method has non-**`void`** return type, it must always return a value.

```
private int thisIsWrong(int x) {
    if (x == 5) {
        return 0;
    }
}
```

What do we return if **`x != 5`**?

Subtleties of **`return`**

- If a method has non-**`void`** return type, it must always return a value.

```
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    } else {
        return 1;
    }
}
```

Many Happy **`returns`**

- A method may have multiple return statements. The method ends as soon as **`return`** is executed.

```
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    } else {
        return 1;
    }
}
```

Many Happy **returns**

- A method may have multiple return statements. The method ends as soon as **return** is executed.

```
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    }
    return 1;
}
```

The only way we can get here is if x is not equal to 5.