JavaScript Graphics

Jerry Cain
CS 106AX
October 2, 2023
slides leveraged from those written by Eric Roberts

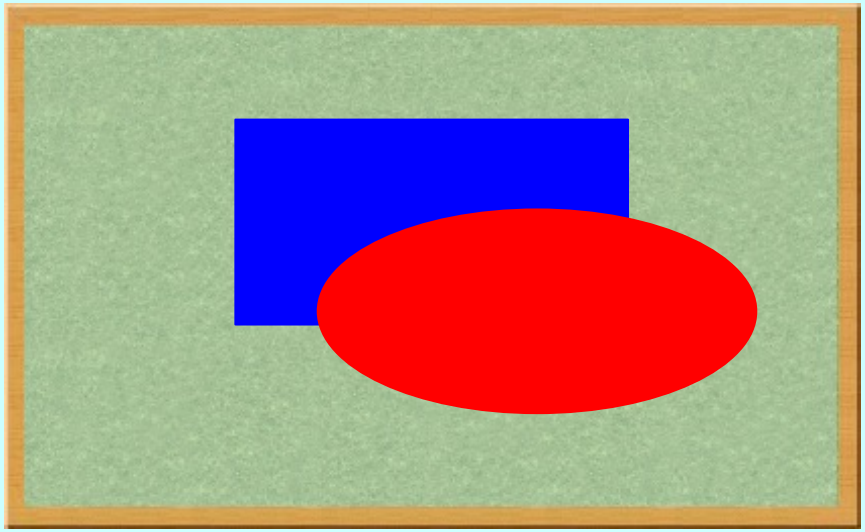
JavaScript Graphics

- In addition to **`JSConsole.js`**, CS 106AX provides a library called **`JSGraphics.js`** that makes it easy to create graphical programs.
- The structure of the **`index.html`** file for graphics programs is like the one used for **`HelloWorld`**. The **`BlueRectangle`** program introduced later uses the following **`index.html`**:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Blue Rectangle</title>
  <script type="text/javascript" src="JSGraphics.js"></script>
  <script type="text/javascript" src="BlueRectangle.js"></script>
 </head>
 <body onload="BlueRectangle()"></body>
</html>
```

The Graphics Model

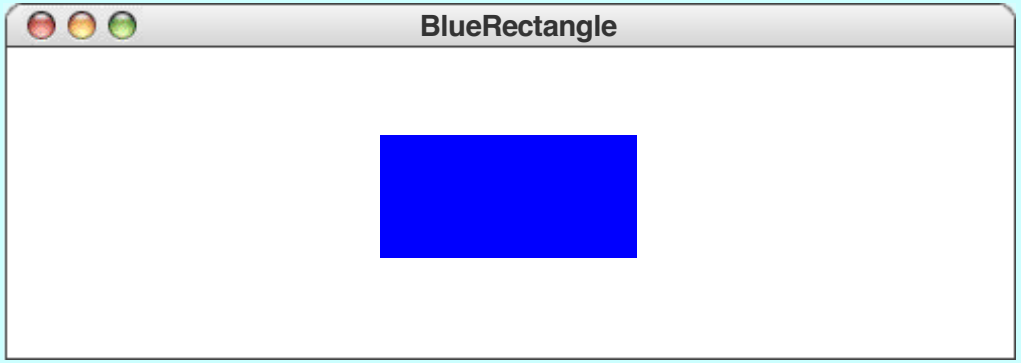
- The **`JSGraphics.js`** library uses a graphics model based on the metaphor of a ***collage***.
- A collage is akin to a child's felt board that serves as a backdrop for colored shapes that stick to the felt surface. As an example, the following diagram illustrates the process of adding a blue rectangle and a red oval to a felt board:



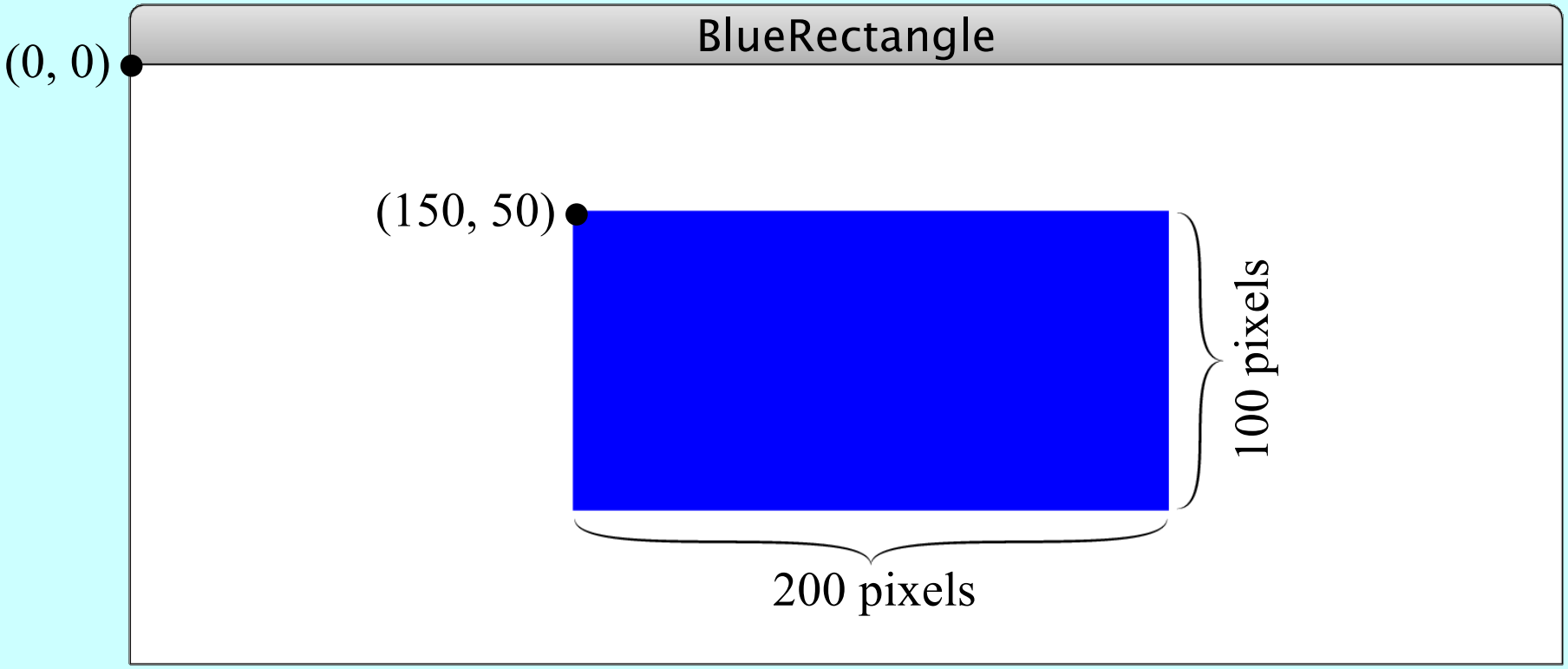
- Note that new objects potentially obscure those added earlier. This layering arrangement is called the ***stacking order***.

The `BlueRectangle` Program

```
function BlueRectangle() {
   let gw = GWindow(500, 200);
   let rect = GRect(150, 50, 200, 100);
   rect.setColor("Blue");
   rect.setFilled(true);
   gw.add(rect);
}
```



The JavaScript Coordinate System



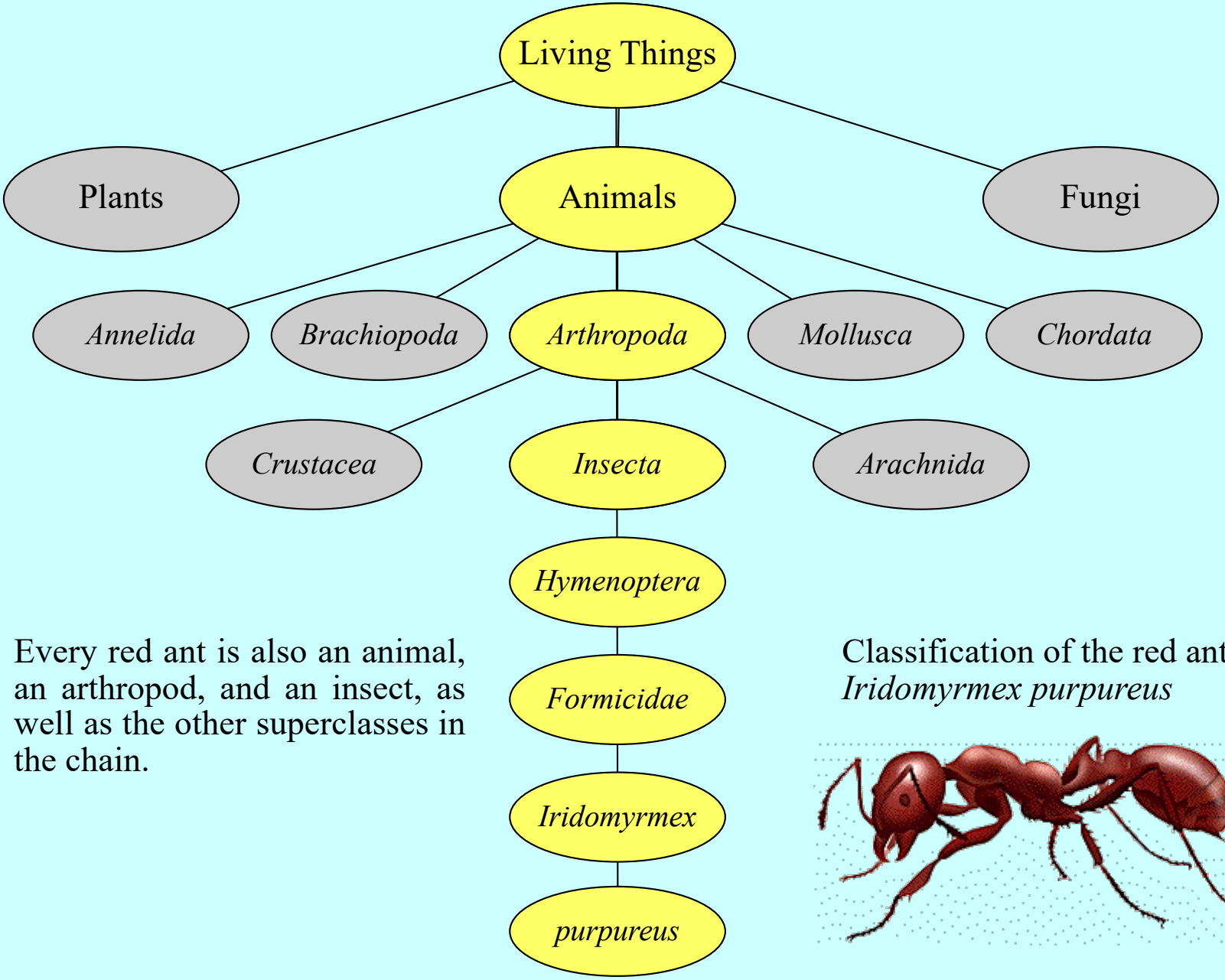
- Positions and distances on the screen are measured in terms of ***pixels***, which are the small dots that cover the screen.
- Unlike traditional mathematics, JavaScript defines the ***origin*** of the coordinate system to be in the upper left corner. Values for the *y* coordinate increase as you move downward.

Systems of Classification

- In the mid-18th century, the Scandinavian botanist Carl Linnaeus revolutionized the study of biology by developing a new system for classifying plants and animals in a way that revealed their structural relationships and paved the way for Darwin's theory of evolution a century later.
- Linnaeus's contribution was to recognize that organisms fit into a hierarchy in which the placement of individual species reflects their anatomical similarities.

Carl Linnaeus (1707–1778)

Biological Class Hierarchy



Every red ant is also an animal, an arthropod, and an insect, as well as the other superclasses in the chain.

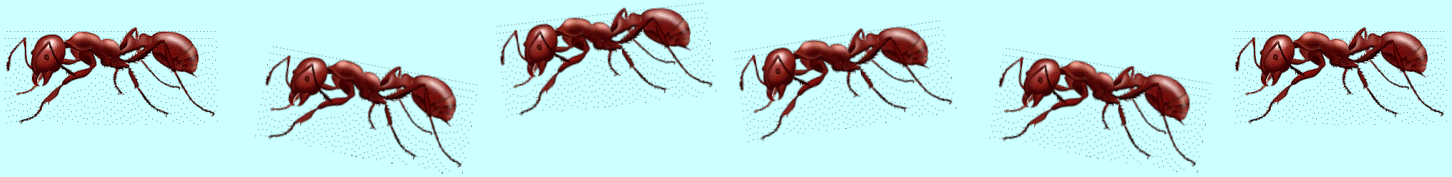
Classification of the red ant *Iridomyrmex purpureus*

Instances *vs.* Patterns

Drawn after you, you pattern of all those.

—William Shakespeare, Sonnet 98

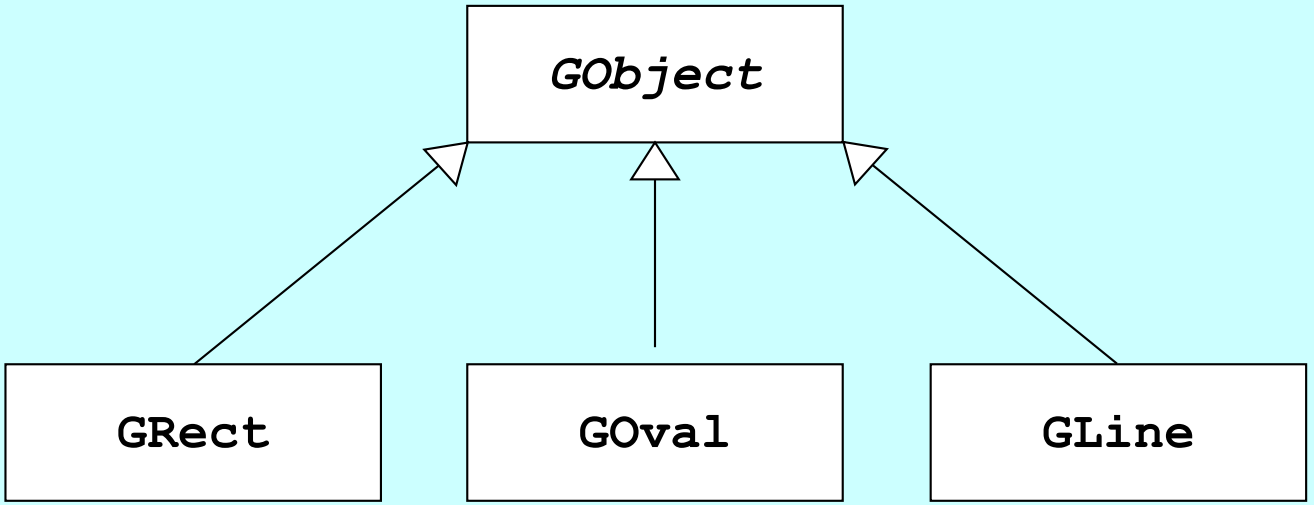
- In thinking about any classification scheme—biological or otherwise—it is important to draw a distinction between a class and specific instances of that class. In the most recent example, the designation *Iridomyrmex purpureus* is not itself an ant, but rather a ***class*** of ant. There can be (and usually are) many ants, each of which is an individual of that class.



- Each of these fire ants is an ***instance*** of a particular class of ants. Each instance is of the species *purpureus,* the genus *Iridomyrmex,* the family *Formicidae* (which makes it an ant), and so on. Thus, each ant is not only an ant, but also an insect, an arthropod, and an animal.

The `GObject` Hierarchy

- The classes that represent graphical objects form a hierarchy, part of which looks like this:



- The **`GObject`** class represents the collection of all graphical objects.
- The three subclasses shown in this diagram correspond to particular types of objects: rectangles, ovals, and lines. Any **`GRect`**, **`GOval`**, or **`GLine`** is also a **`GObject`**.

Creating a `GWindow` Object

- The first step in writing a graphical program is to create a window using the following function declaration, where *width* and *height* indicate the size of the window:

```
let gw = GWindow(width, height);
```

- The following operations work with any `GWindow` object:

`gw.add(`*object***`)`** Adds an object to the window.
`gw.add(`*object***`, `***x***`, `***y***`)`** Adds an object to the window after first moving it to (*x*, *y*).
`gw.remove(`*object***`)`** Removes the object from the window.
`gw.getWidth()` Returns the width of the graphics window in pixels.
`gw.getHeight()` Returns the height of the graphics window in pixels.

Operations on the `GObject` Class

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

object`.getX()` Returns the *x* coordinate of this object.
object`.getY()` Returns the *y* coordinate of this object.
object`.getWidth()` Returns the width of this object.
object`.getHeight()` Returns the height of this object.
object`.setColor(`*color*`)` Sets the color of the object to the specified color.

- All coordinates and distances are measured in pixels.
- Each color is a string, such as **`"Red"`** or **`"White"`**. The names of the standard colors are defined in Figure 4-5 on page 125.

Drawing Geometrical Objects

Functions to create geometrical objects:

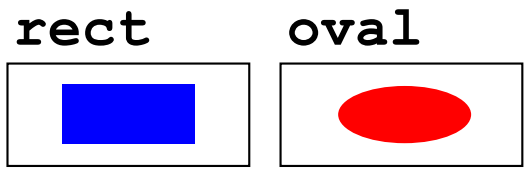
`GRect(`*x*`, `*y*`, `*width*`, `*height*`)`** Creates a rectangle whose upper left corner is at (x, y) of the specified size.
`GOval(`*x*`, `*y*`, `*width*`, `*height*`)`** Creates an oval that fits within a rectangle with the same dimensions.
`GLine(`x_0**`, `**y_0**`, `**x_1**`, `**y_1**`)`** Creates a line extending from (x_0, y_0) to (x_1, y_1).

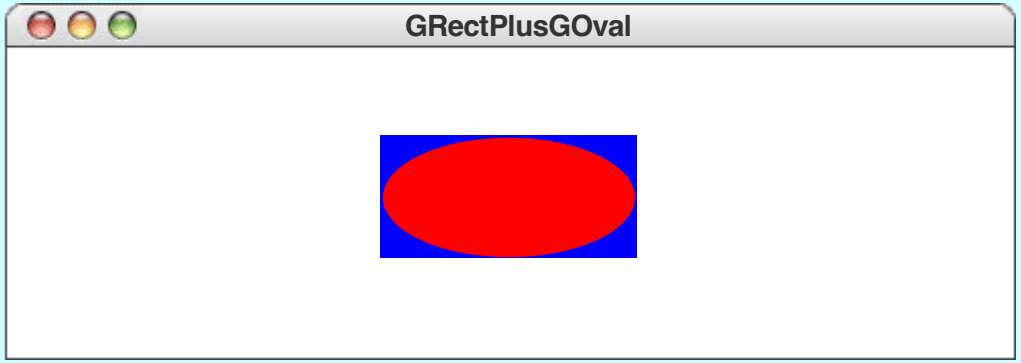
Methods shared by the **`GRect`** and **`GOval`** classes:

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.

The GRectPlusGOval Program

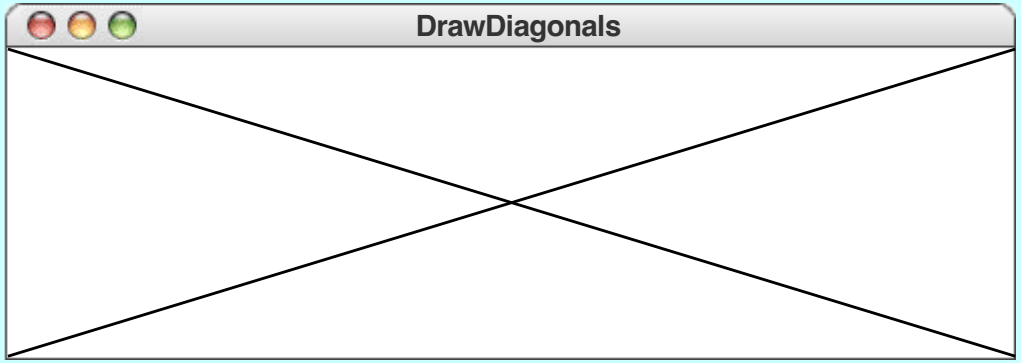
```
function GRectPlusGOval() {
   let gw = GWindow(500, 200);
   let rect = GRect(150, 50, 200, 100);
   rect.setFilled(true);
   rect.setColor("Blue");
   gw.add(rect);
   let oval = GOval(150, 50, 200, 100);
   oval.setFilled(true);
   oval.setColor("Red");
   gw.add(oval);
}
```





The DrawDiagonals Program

```
/* Constants */

const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 200;

function DrawDiagonals() {
   let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   gw.add(GLine(0, 0, GWINDOW_WIDTH, GWINDOW_HEIGHT));
   gw.add(GLine(0, GWINDOW_HEIGHT, GWINDOW_WIDTH, 0));
}
```



The Checkerboard Program

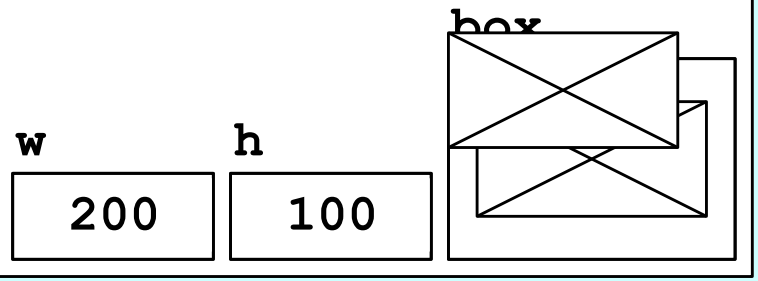
```
const GWINDOW_WIDTH = 500;  /* Width of the graphics window  */
const GWINDOW_HEIGHT = 300; /* Height of the graphics window */
const N_COLUMNS = 8;        /* Number of columns             */
const N_ROWS = COLUMNS;     /* Number of rows                */
const SQUARE_SIZE = 35;     /* Size of a square in pixels    */

function Checkerboard() {
   let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   let x0 = (gw.getWidth() - N_COLUMNS * SQUARE_SIZE) / 2;
   let y0 = (gw.getHeight() - N_ROWS * SQUARE_SIZE) / 2;

   for (let i = 0; i < N_ROWS; i++) {
      for (let j = 0; j < N_COLUMNS; j++) {
         let x = x0 + j * SQUARE_SIZE;
         let y = y0 + i * SQUARE_SIZE;
         let sq = GRect(x, y, SQUARE_SIZE, SQUARE_SIZE);
         let filled = (i + j) % 2 !== 0;
         sq.setFilled(filled);
         gw.add(sq);
      }
   }
}
```

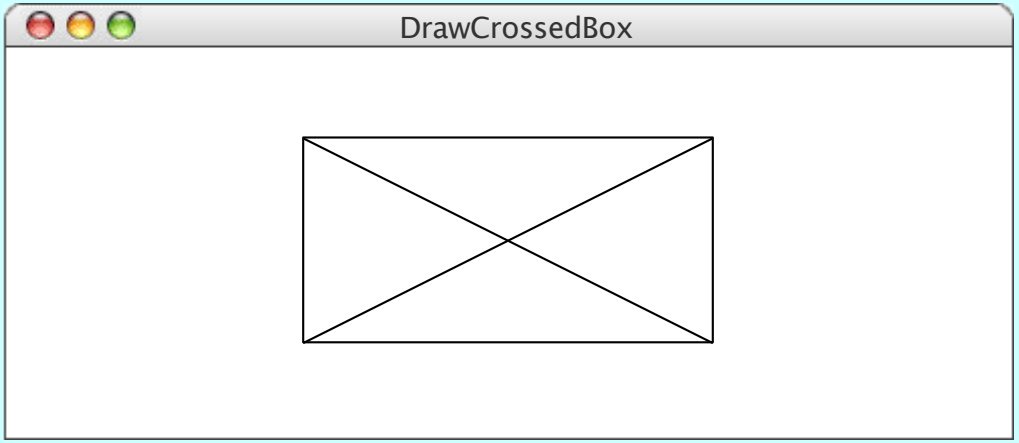
Creating Compound Objects

- The **`GCompound`** class makes it possible to combine and layer several graphical objects so that the resulting structure behaves as a single **`GObject`**.
- The easiest way to think about the **`GCompound`** class is as a combination of a **`GWindow`** and a **`GObject`**. A **`GCompound`** is like a **`GWindow`** in that you can add objects to it, but it is also like a **`GObject`** in that you can add it to a graphics window.
- A **`GCompound`** object has its own coordinate system that is expressed relative to a reference point that you invent. When you add new objects to the **`GCompound`**, you use the local coordinate system based on that reference point. When you add the **`GCompound`** to the graphics window, all you have to do is set the location of the reference point, and then the individual components will automatically appear in the right locations relative to that point.

Using a GCompound class

```
function DrawCrossedBox() {
  function createCrossedBox(w, h) {
    let box = GCompound();
    box.add(GRect(-w / 2, -h / 2, w, h));
    box.add(GLine(-w / 2, -h / 2, w / 2, h / 2));
    box.add(GLine(-w / 2, h / 2, w / 2, -h / 2));
    return box;
  }
```





The End