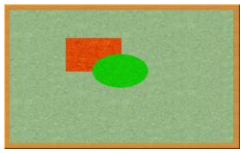
Revisiting `acm.graphics`

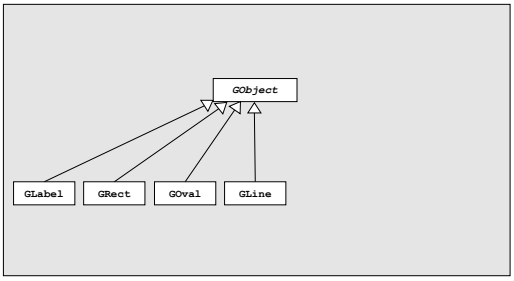
- **collage** model
  - create image by adding objects to a canvas



- Newer objects obscure those added earlier
- Layering is called the **stacking order** (or *z*-order)

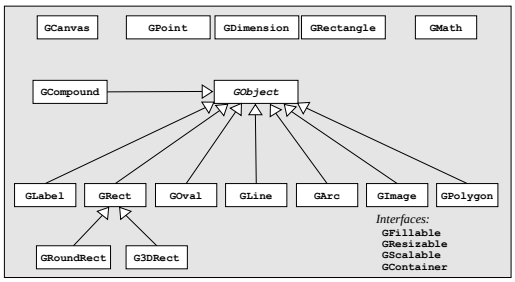
Using portions of slides by Eric Roberts

Structure of `acm.graphics` Package



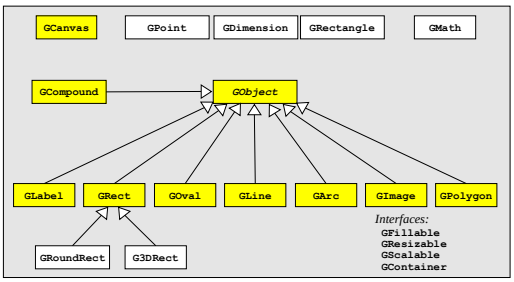
Using portions of slides by Eric Roberts

Structure of `acm.graphics` Package



Using portions of slides by Eric Roberts

Structure of `acm.graphics` Package



Using portions of slides by Eric Roberts

`GCanvas`

- Used to represent background canvas of collage
- **`GraphicsProgram`** automatically creates **`GCanvas`** that fills the entire program window
- When you call **`add(...)`** in **`GraphicsProgram`**, it is *forwarding* your call to the **`GCanvas`**
  - Forwarding is just when receiver of message then calls some other object with that same message

Using portions of slides by Eric Roberts

Methods in `GCanvas` and `GraphicsProgram`

The following methods are available in both the **`GCanvas`** and **`GraphicsProgram`** classes:

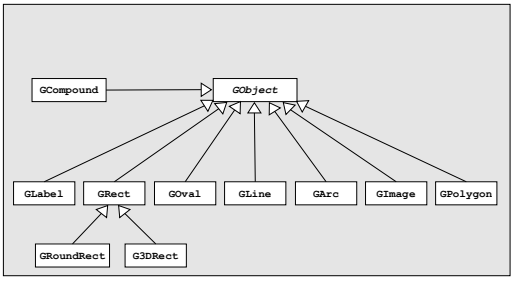
`add`(*object*)	Adds the object to the canvas at the front of the stack
`add`(*object*, *x*, *y*)	Moves the object to (*x*, *y*) and then adds it to the canvas
`remove`(*object*)	Removes the object from the canvas
`removeAll`()	Removes all objects from the canvas
`getElementAt`(*x*, *y*)	Returns the frontmost GObject at (*x*, *y*), or **`null`** if none
`getWidth`()	Returns the width in pixels of the entire canvas
`getHeight`()	Returns the height in pixels of the entire canvas
`setBackground`(*c*)	Sets the background color of the canvas to *c*.

The following methods are available in **`GraphicsProgram`** only:

`pause`(*milliseconds*)	Pauses the program for the specified time in milliseconds
`waitForClick`()	Suspends the program until the user clicks the mouse

Using portions of slides by Eric Roberts

Class hierarchy of `GObject`



Using portions of slides by Eric Roberts

Methods Common to All `GObject`s

`setLocation``(`*x*`, `*y*`)`	Resets the location of the object to the specified point
`move``(`*dx*`, `*dy*`)`	Moves the object *dx* and *dy* pixels from its current position
`getX()`	Returns the *x* coordinate of the object
`getY()`	Returns the *y* coordinate of the object
`getWidth()`	Returns the horizontal width of the object in pixels
`getHeight()`	Returns the vertical height of the object in pixels
`contains``(`*x*`, `*y*`)`	Returns **`true`** if the object contains the specified point
`setColor``(`*c*`)`	Sets the color of the object to the **`Color`** *c*
`getColor()`	Returns the color currently assigned to the object
`setVisible``(`*flag*`)`	Sets the visibility flag (**`false`** = invisible, **`true`** = visible)
`isVisible()`	Returns **`true`** if the object is visible
`sendToFront()`	Sends the object to the front of the stacking order
`sendToBack()`	Sends the object to the back of the stacking order
`sendForward()`	Sends the object forward one position in the stacking order
`sendBackward()`	Sends the object backward one position in the stacking order

Using portions of slides by Eric Roberts

Methods Defined by Interfaces

`GFillable` (**`GRect`**, **`GOval`**, **`GArc`**, **`GPolygon`**)

`setFilled``(`*flag*`)`	Sets the fill state for the object (**`false`** = outlined, **`true`** = filled)
`isFilled()`	Returns the fill state for the object
`setFillColor``(`*c*`)`	Sets the color used to fill the interior of the object to *c*
`getFillColor()`	Returns the fill color

`GResizable` (**`GOval`**, **`GRect`**, **`GImage`**)

`setSize``(`*width*`, `*height*`)`	Sets the dimensions of the object as specified
`setBounds``(`*x*`, `*y*`, `*width*`, `*height*`)`	Sets the location and dimensions together

`GScalable` (**`GLine`**, **`GOval`**, **`GRect`**, **`GArc`**, **`GCompound`**, **`GImage`**, **`GPolygon`**,)

`scale``(`*sf*`)`	Scales both dimensions of the object by *sf*
`scale``(`*sx*`, `*sy*`)`	Scales the object by *sx* horizontally and *sy* vertically

Using portions of slides by Eric Roberts

A little animation demo: `BouncingBall.java`

The `GLabel` Class

```
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);
   }
}
```

Using portions of slides by Eric Roberts

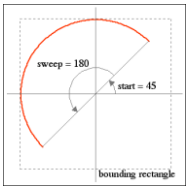
The Geometry of the `GLabel` Class

- The **`GLabel`** class typesetting concepts:
  - **baseline**: imaginary line on which the characters rest.
  - **origin**: point on the baseline at which the label begins.
  - **height** (of font): distance between successive baselines.
  - **ascent**: distance characters rise above the baseline.
  - **descent**: distance characters drop below the baseline.

Using portions of slides by Eric Roberts

The `GArc` Class

- **`GArc`** – arc formed by taking section from perimeter of oval.
- Conceptually, steps necessary to define an arc are:
  - Specify the coordinates and size of the bounding rectangle
  - Specify **start angle** (angle at which the arc begins)
  - Specify **sweep angle** (how far the arc extends)
- Angles measured in degrees starting at the +*x* axis (the 3:00 o'clock position) and increasing **counterclockwise**.
- Negative values for the *start* and *sweep* angles signify a clockwise direction.

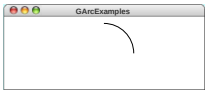


`GArc` Geometry

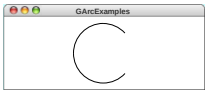
Assume: **`cx`** and **`cy`** are coordinates of window center

`d` (diameter) is 0.8 times the screen height.

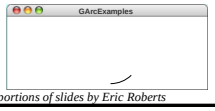
```
GArc a1 = new GArc(d, d, 0, 90);
add(a1, cx - d / 2, cy - d / 2);
```



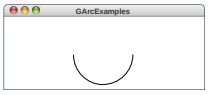

```
GArc a2 = new GArc(d, d, 45, 270);
add(a2, cx - d / 2, cy - d / 2);
```




```
GArc a3 = new GArc(d, d, -90, 45);
add(a3, cx - d / 2, cy - d / 2);
```




```
GArc a4 = new GArc(d, d, 0, -180);
add(a4, cx - d / 2, cy - d / 2);
```



Filled Arcs

- **`GArc`** class implements **`GFillable`** interface
- Filled **`GArc`** is the pie-shaped wedge formed by the center and the endpoints of the arc

```
public void run() {
   GArc arc = new GArc(0, 0, getWidth(), getHeight(),
                       0, 90);
   arc.setFilled(true);
   add(arc);
}
```

The `GImage` Class

- **`GImage`** class is used to display an image from a file

`new GImage(`*image file*`,` *x*`,` *y*`)`**

image file: name of a file containing image

x and *y*: coordinates of upper left corner of image

- Looks for file in current project directory and then in a subdirectory named **`images`**.
- GIF (**`.gif`**) and JPEG (**`.jpg`** or **`.jpeg`**) supported

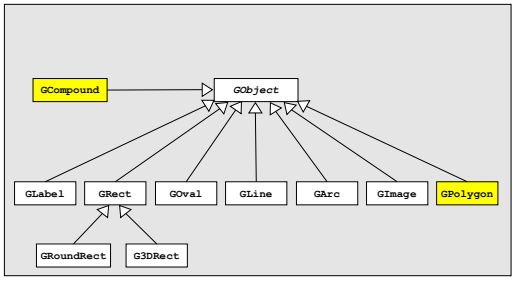
Example of the `GImage` Class

```
public void run() {
  GImage image = new GImage("StanfordSeal.gif");
  add(image, 0, 0);
}
```

Resizing `GImage`s

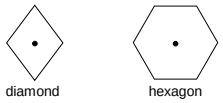
```
public void run() {
  GImage image = new GImage("StanfordSeal.gif");
  image.scale(1.5, 0.5);
  add(image, 0, 0);
}
```

Class hierarchy of **GObject**



The **GPolygon** Class

- **GPolygon**: represent graphical objects bound by line segments.



- A **GPolygon** has a **reference point** that is convenient for that particular shape
- Position the vertices relative to that reference point.
- Convenient reference point is often center of object.

Constructing a **GPolygon** Object

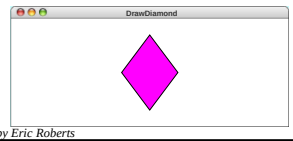
- Create an empty polygon
- Add vertices one at a time using **addVertex**(*x*, *y*)
  - *x* and *y* relative to **reference point** of polygon
- After setting initial vertex using **addVertex**(*x*, *y*), can add remaining ones using:
  - **addVertex**(*x*, *y*) adds a new vertex relative to the reference point
  - **addEdge**(*dx*, *dy*) adds a new vertex relative to the preceding one
- Polygon "closed" for you
  - automatically attaches first and last vertices

Drawing a Diamond (**addVertex**)

The following program draws a diamond using **addVertex**:

```
public void run() {
private GPolygon createDiamond(double width, double height) {
   GPolygon diamond = new GPolygon();
   diamond.addVertex(-width / 2, 0);
   diamond.addVertex(0, -height / 2);
   diamond.addVertex(width / 2, 0);
   diamond.addVertex(0, height / 2);
   return diamond;
}
```

diamond



skip simulation

The **GCompound** Class

- **GCompound** allows for combining several graphics objects so they behave like one **GObject**
- Add objects to a **GCompound** (like it was a canvas)
- You can treat whole **GCompound** as one object
- Similar to **GPolygon**, a **GCompound** has a reference point that all objects are added with respect to
- When **GCompound** is added to canvas, it is placed relative to its reference point
- Let's draw a face: