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 \texttt{acm.graphics} Package

- \texttt{GCanvas}
- \texttt{GPoint}
- \texttt{GDimension}
- \texttt{GRectangle}
- \texttt{GMath}

\begin{itemize}
  \item \texttt{GObject}
  \item \texttt{GCompound}
  \item \texttt{GLabel}
  \item \texttt{GRect}
  \item \texttt{GOval}
  \item \texttt{GLine}
  \item \texttt{GArc}
  \item \texttt{GImage}
  \item \texttt{GPolygon}
  \item \texttt{GRoundRect}
  \item \texttt{G3DRect}
\end{itemize}

Interfaces:
- \texttt{GFillable}
- \texttt{GResizable}
- \texttt{GScalable}
- \texttt{GContainer}

\textit{Using portions of slides by Eric Roberts}
Structure of `acm.graphics` Package

- `GCanvas`
- `GPoint`
- `GDimension`
- `GRectangle`
- `GMath`
- `GObject`
- `GCompound`
- `GLabel`
- `GRect`
- `GOval`
- `GLine`
- `GArc`
- `GImage`
- `GPolygon`
- `GRoundRect`
- `G3DRect`

Interfaces:
- `GFillable`
- `GResizable`
- `GScalable`
- `GContainer`

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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(object)</code></td>
<td>Adds the object to the canvas at the front of the stack</td>
</tr>
<tr>
<td><code>add(object, x, y)</code></td>
<td>Moves the object to ((x, y)) and then adds it to the canvas</td>
</tr>
<tr>
<td><code>remove(object)</code></td>
<td>Removes the object from the canvas</td>
</tr>
<tr>
<td><code>removeAll()</code></td>
<td>Removes all objects from the canvas</td>
</tr>
<tr>
<td><code>getElementAt(x, y)</code></td>
<td>Returns the frontmost GObject at ((x, y)), or <code>null</code> if none</td>
</tr>
<tr>
<td><code>getWidth()</code></td>
<td>Returns the width in pixels of the entire canvas</td>
</tr>
<tr>
<td><code>getHeight()</code></td>
<td>Returns the height in pixels of the entire canvas</td>
</tr>
<tr>
<td><code>setBackground(c)</code></td>
<td>Sets the background color of the canvas to (c).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pause(milliseconds)</code></td>
<td>Pauses the program for the specified time in milliseconds</td>
</tr>
<tr>
<td><code>waitForClick()</code></td>
<td>Suspends the program until the user clicks the mouse</td>
</tr>
</tbody>
</table>

Using portions of slides by Eric Roberts
## Methods Common to All **GObjects**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setLocation ((x, y))</td>
<td>Resets the location of the object to the specified point</td>
</tr>
<tr>
<td>move ((dx, dy))</td>
<td>Moves the object (dx) and (dy) pixels from its current position</td>
</tr>
<tr>
<td>getX()</td>
<td>Returns the (x) coordinate of the object</td>
</tr>
<tr>
<td>getY()</td>
<td>Returns the (y) coordinate of the object</td>
</tr>
<tr>
<td>getWidth()</td>
<td>Returns the horizontal width of the object in pixels</td>
</tr>
<tr>
<td>getHeight()</td>
<td>Returns the vertical height of the object in pixels</td>
</tr>
<tr>
<td>contains ((x, y))</td>
<td>Returns \textbf{true} if the object contains the specified point</td>
</tr>
<tr>
<td>setColor (c)</td>
<td>Sets the color of the object to the \texttt{Color} (c)</td>
</tr>
<tr>
<td>getColor()</td>
<td>Returns the color currently assigned to the object</td>
</tr>
<tr>
<td>setVisible (flag)</td>
<td>Sets the visibility flag (false=invisible, true=visible)</td>
</tr>
<tr>
<td>isVisible()</td>
<td>Returns \textbf{true} if the object is visible</td>
</tr>
<tr>
<td>sendToFront()</td>
<td>Sends the object to the front of the stacking order</td>
</tr>
<tr>
<td>sendToBack()</td>
<td>Sends the object to the back of the stacking order</td>
</tr>
<tr>
<td>sendForward()</td>
<td>Sends the object forward one position in the stacking order</td>
</tr>
<tr>
<td>sendBackward()</td>
<td>Sends the object backward one position in the stacking order</td>
</tr>
</tbody>
</table>

*Using portions of slides by Eric Roberts*
# Methods Defined by Interfaces

## GFillable (GRect, GOval, GArc, GPolygon)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setFilled(flag)</td>
<td>Sets the fill state for the object (false=outlined, true=filled)</td>
</tr>
<tr>
<td>isFilled()</td>
<td>Returns the fill state for the object</td>
</tr>
<tr>
<td>setFillColor(c)</td>
<td>Sets the color used to fill the interior of the object to c</td>
</tr>
<tr>
<td>getFillColor()</td>
<td>Returns the fill color</td>
</tr>
</tbody>
</table>

## GResizable (GOval, GRect, GImage)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setSize(width, height)</td>
<td>Sets the dimensions of the object as specified</td>
</tr>
<tr>
<td>setBounds(x, y, width, height)</td>
<td>Sets the location and dimensions together</td>
</tr>
</tbody>
</table>

## GScalable (GLine, GOval, GRect, GArc, GCompound, GImage, GPolygon,)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale(sf)</td>
<td>Scales both dimensions of the object by sf</td>
</tr>
<tr>
<td>scale(sx, sy)</td>
<td>Scales the object by sx horizontally and sy vertically</td>
</tr>
</tbody>
</table>
A little animation demo:
BouncingBall.java
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

- **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.

*Using portions of slides by Eric Roberts*
GArC Geometry

Assume: \( \textbf{cx} \) and \( \textbf{cy} \) are coordinates of window center
\( \textbf{d} \) (diameter) is 0.8 times the screen height.

```java
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);
```

Using portions of slides by Eric Roberts
Filled Arcs

- **GArc** class implements **GFillable** interface

- Filled **GArc** is the pie-shaped wedge formed by the center and the endpoints of the arc

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

Using portions of slides by Eric Roberts
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

*Using portions of slides by Eric Roberts*
Example of the **GImage** Class

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

Using portions of slides by Eric Roberts
Resizing **GImages**

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

*Using portions of slides by Eric Roberts*
Class hierarchy of GObject

Using portions of slides by Eric Roberts
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.

*Using portions of slides by Eric Roberts*
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

Using portions of slides by Eric Roberts
Drawing a Diamond (**addVertex**)

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

```java
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;
    }
```

Using portions of slides by Eric Roberts
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:

Using portions of slides by Eric Roberts