Module IV: Introduction to Drawing

Learning Objectives

- How to draw simple shapes on an image.
- How to set the color to draw with
- How to set the font to write with
- How to draw text on top of an image

You will learn how to us the Graphics object to draw on a picture.

We already learned a lot about how to create and manipulate images in various ways in Java. Now we will look at a class called java.awt.Graphics, which allows us to draw lines, shapes and text on an image. To use this class, you'll need to import java.awt.Graphics or import java.awt.*

Basic drawing

Drawing on a picture is really just setting pixels to the color we want. For example, we can draw a horizontal line across a picture by setting all of the pixels in one row of the picture to the color black. If we want to grid of black lines on a picture, we just need to determine which rows and columns make up the grid and set all those to the color black.

We could add vertical lines to a picture every 20 pixels, by setting up our nested loops like this:

```
Start with x = 20, x < width, x += 20
        Start with y = 0, y < height, y++
        ...
```

For horizontal lines:

```
Start with x = 0, x < width, x++
        Start with y = 20, y < height, y+=20
        ...
```

Note... The notation += is shorthand for **variableName** = **variableName** + **number**. Similarly, -= is shorthand for subtraction.

learn by doing

Write a method drawGrid in your Picture class to add horizontal and vertical lines to a picture every 20 pixels.

Test your method using:

```
Picture p = new Picture (“<file location>”);
p.drawGrid();
p.show();
```

solution:

```
public void drawGrid() {
   for (int x = 20; x < this.getWidth(); x+=20) {
      for (int y = 0; y < this.getHeight(); y++) {
         this.getPixel(x,y).setColor(Color.BLACK);
      }
   }
   for (int x = 0; x < this.getWidth(); x++) {
      for (int y = 20; y < this.getHeight(); y+=20) {
         this.getPixel(x,y).setColor(Color.BLACK);
      }
   }
}
```

Java Graphics

You can use the same basic idea (changing the colors of specific pixels) to draw a circle, a string of characters, lines, or other shapes on an image. But determining which pixels to change would require some thought. Since circles, strings, lines, and other shapes are basic elements of almost any on-screen drawing, good programming practice dictates that we don't ask every programmer to re-invent the wheel. Instead, Java provides the Graphics class to do this sort of stuff.

Every picture object has a Graphics object associated with it, which has methods for drawing simple lines, shapes, and text. The method for retrieving the Graphics object is:

pictureObj.getGraphics()

The methods in the **java.awt.Graphics** class let you paint on an image as follows:

- Pick any color to draw with.
- Draw shapes: circles, ellipses, rectangles, lines, arcs, polygons
- Shapes drawn will cover whatever is on the image already, including other drawings.
- Draw text: Set the font and write text strings in that font.

Set the current color using:

graphicsObj.setColor(Color c);

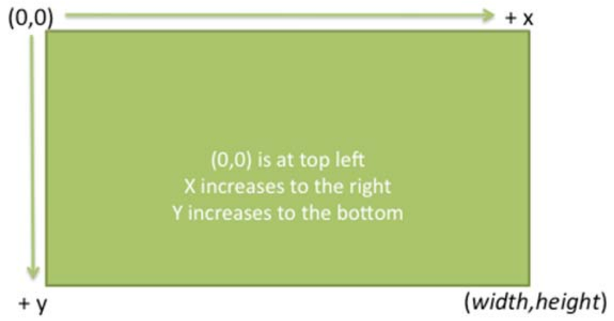
Similarly, you can get the current color using:

graphicsObj.getColor();

As described when we first began manipulating images, colors can be created from scratch using new Color(**red, green, blue**), referenced using pre-defined constants (such as Color.RED), and so on. Make sure that you import **java.awt.*** or **java.awt.Color** before you use colors, so the class is defined.

Graphics Environment

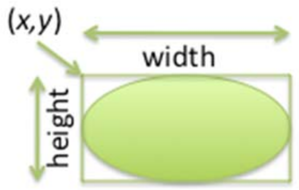
As we have discussed, the upper left corner of a **Picture** is at **(0,0)**, while the bottom right corner is at **(width, height)**. Every Graphics object in Java functions the same way: Any commands you give it assume that the ***x*** and ***y*** coordinates of a point are counted from **(0,0)** at top left of the image, something like this:



Drawing Circles and Ovals

To draw a circle or oval, we can use

drawOval(x,y,width,height) or **fillOval(x,y,width,height)**



x and *y* are the coordinates of the top left corner of the enclosing rectangle, and *width* and *height* are the width and height of the enclosing rectangle. The oval is drawn so it just touches the edges of this enclosing rectangle. If *width* and *height* are the same value, the result is a circle instead of an oval.

fillOval(x,y,width,height) does the same thing as **drawOval(...)**, except it fills in the inside of the oval, instead of just drawing the outside of it.

learn by doing

As an exercise, try writing a method to add a yellow sun to the image "beach.jpg". Call your method drawSun(). Test your method on beach.jpg.

Solution

```
public void drawSun(){
   Graphics g = this.getGraphics();
   g.setColor(Color.YELLOW);
   g.fillOval(155,60,20,20);

}
```

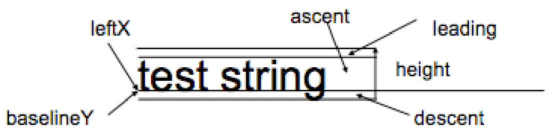
Note... If you want, you can save your new picture to disk using: pictureObj.write("<path>smallerPict.jpg");

Fonts and Strings

One of the most common things drawn on screen in today's interactive systems is text. Text is drawn using a method call drawString(). In addition to the color of text, you can also control its font.

- Define a font. First import java.awt.Font Then use Font labelFont = new Font("TimesRoman", Font.BOLD, 24); to define a bold Times Roman font or Font labelFont = new Font("TimesRoman", Font.PLAIN, 24); for a normal Helvetica font. Allowed styles are PLAIN, BOLD, ITALIC, and BOLDITALIC. These are all available as fields in the Font class, just as the Color class provides fields such as BLACK and RED.
- Tell the graphics object to use this font: graphicsObj.setFont(labelFont)
- Now I can use the graphics object to get font information: graphicsObj.getStyle() to get font style graphicsObj.getSize() for the point size, graphicsObj.getName() for the font name (e.g. "TimesRoman", passed in the constructor) and graphicsObj.getFamily() for the font family (e.g. Helvetica).

The graphics object provides a method to draw a string on an image, which looks like graphicsObj.drawString("string", leftX, baselineY). Just as with anything you draw on screen, the Graphics object needs to know the location of a piece of text (it can get the size by knowing the font). The location coordinates are defined as follows:



leftX simply refers to the leftmost part of the string. *baselineY* is more complex: It refers to the Y position of the base of characters that do not have a descender. For example, the 'g' in "test string" descends below *baselineY*.

We can get the font information by using the FontMetrics class.

```
import java.awt.FontMetrics;
FontMetrics currFontMetrics = graphicsObj.getFontMetrics();
int baselineY = currFontMetrics.getHeight() - currFontMetrics.getDescent();
int width = currFontMetrics.stringWidth("string");
```

Note that width isn't required to draw a string, but can be useful regardless.

learn by doing

LOLCATs are joke images that show pictures of animals (usually cats) with a funny caption (often as if the cat were speaking), a website dedicated to lolcats can be found here: http://icanhas.cheezburger.com/

Write a method drawstring that adds a caption to a picture. Please be sure to include a comment indicating what string you decided to add, that's half the fun after all :)! Your method should:

Take a string as a parameter
Set the color to draw the text with
Set the font to use for the string.
Draw the string.

Solution

```
/**
 * Method to draw a string near the bottom left of a picture. Nearness is
 * defined using a percentage of the pictures width. (e.g if a picture is
 * 100x200 in size, a 10% offset would place the string at (10, 190).
 * @param howNear a percentage (0..1) defining how much to offset the string from (0, height).
 */
public void drawString(String text, double howNear) {
   int leftX, baselineY, offset;
   //get the graphics object
   Graphics g = this.getGraphics();
   g.setColor(Color.YELLOW);
   g.setFont(new Font("Helvetica", Font.BOLD, 24));
   //calculate the pixel offset
   offset = (int) (this.getWidth() * howNear);
   //calculate the left x position and baseline
   leftX = offset;
   baselineY = this.getHeight() - offset;
   //draw the string
   g.drawString(text, leftX, baselineY);
}
```

learn by doing

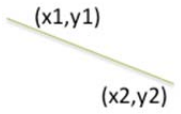
Try to make a more general method drawString(String, baselineY) and centers the string in x.

Solution:

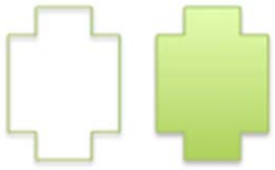
```
public void drawString(String text, int baselineY) {
   int leftX, stringWidth, pictureMiddleX;
   //get the graphics object
   Graphics g = this.getGraphics();
   g.setColor(Color.YELLOW);
   g.setFont(new Font("Helvetica", Font.BOLD, 24));
   //calculate the pixel x location
   FontMetrics currFontMetrics = g.getFontMetrics();
   // get the width of the string
   stringWidth = currFontMetrics.stringWidth(text);
   // get the middle of the picture
   pictureMiddleX = this.getWidth() / 2;
   // the middle of the string should align with the middle of the picture.
  // To do this we have to place the left edge of the string at pictureMiddleX - stringWidth/2
   leftX = pictureMiddleX - stringWidth / 2;
   g.drawString(text, leftX, baselineY);
}
```

Other types of drawing elements:

We can draw a line on a picture by giving its coordinates with graphicsObj.drawLine(x1,y1,x2,y2);



For the non-mathematically inclined, a polygon is just a set of points with lines connecting them to form a closed path: rectangles, trapezoids, pentagons and triangles are just a few examples. We can draw a polygon given a set of all it's vertices (the endpoints of each line). Once you have calculated the (x, y) location of each vertex, you will need to place all of the x locations (in clockwise or counterclockwise order) in an array called xArray and all the y locations (in clockwise or counterclockwise order) in an array called **yArray**. Use **graphicsObj.drawPolygon(xArray, yArray, numPoints);** to draw an outline, and **graphicsObj.fillPolygon(xArray, yArray, numPoints);** for a filled polygon.



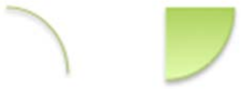
Outlined and Filled Arcs:

Outlined Arc

graphicsObj.drawArc(topLeftX, topLeftY, width, height, startAngle, arcAngle);

Filled Arc

graphicsObj.fillArc(topLeftX, topLeftY, width, height, startAngle, arcAngle);



Outlined and Filled Rectangles:

Outlined Rectangle

graphicsObj.drawRect(topLeftX, topLeftY, width, height);

Filled Rectangle

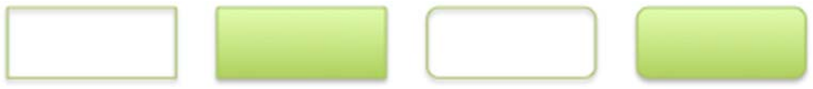
graphicsObj.fillRect(topLeftX,topLeftY,width,height);

Outlined Rounded Rectangle

graphicsObj.drawRoundRect(topLeftX,topLeftY,width,height,arcWidth,arcHeight);

Filled Rounded Rectangle

graphicsObj.fillRoundRect(topLeftX,topLeftY,width,height,arcWidth,arcHeight);



Now that we've covered a number of primitive drawing functions, you can begin to create your own pictures. It's sometimes nice to start with a blank canvas. As described before, you can make pictures from the blank files we provide (lines 1 and 2). You can also create a blank picture using the syntax shown in line 3.

1. Picture pictureObj = new Picture("640x480.jpg");
2. Picture pictureObj = new Picture("7inx95in.jpg");
3. Picture blankPicture = new Picture(width, height);

learn by doing

As an exercise, try to create a method that will draw a simple picture

Use at least one rectangle
Use at least one polygon
Use at least one oval
Use at least one arc

Summary

You can draw by changing the color of the pixels directly. Or you can use java.awt.Graphics to do drawing

- import java.awt.Graphics
- Get a graphics object from a picture object with: Graphics graphics = pictureObj.getGraphics();
- Set the color to draw with: graphics.setColor(color);
- Do simple drawing.
    - graphics.drawLine(x1, y1, x2, y2);
    - graphics.drawOval(x, y, width, height);
    - graphics.drawString("string", offsetX, baselineY);
        - import java.awt.Font
        - graphicsObj.setFont(labelFont); to set the font.
        - gObj.getStyle(), g.getSize(), g.getName(), g.getFamily() gets some font information.
        - Use FontMetrics class for accessing detailed font information
            - import java.awt.FontMetrics
            - FontMetrics currFontMetrics = graphicsObj.getFontMetrics();
            - int baselineY = currFontMetrics.getHeight() - currFontMetrics.getDescent();
            - int width = currFontMetrics.stringWidth("**string**");
    - graphicsObj.drawPolygon(xArray, yArray, numPoints);
    - graphicsObj.drawArc(topLeftX, topLeftY, width, height, startAngle, arcAngle);
    - graphicsObj.drawRect(topLeftX, topLeftY, width, height);
    - graphicsObj.drawRoundRect(topLeftX,topLeftY,width,height,arcWidth,arcHeight);
- Draw a filled in version of everything (but line) using fill*object*