CMSC 113: Computer Science I
Drawing Shapes

There are three basic shapes we learn to draw in Java: lines, ovals, and rectangles. We also learn how to write a text message to the screen. This handout is a useful reference on how to use these shapes in a Java program.

With all shapes, you must remember to `add` the shape after creating it with `new`.

To create a line:

Use a line of code like the following:

```
GLine harry = new GLine(10, 20, 30, 40);
```

Here, we have created a line named `harry` that runs from the point (10, 20) to the point (30, 40). The four numbers in the parentheses are as follows:

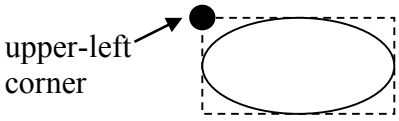
The first number is the *x*-coordinate of the starting point of the line.
The second number is the *y*-coordinate of the starting point of the line.
The third number is the *x*-coordinate of the ending point of the line.
The fourth number is the *y*-coordinate of the ending point of the line.

To create an oval:

Use a line of code like the following:

```
GOval hermione = new GOval(10, 20, 30, 40);
```

Here, we have created an oval named `hermione` that has its upper-left corner at the point (10, 20), is 30 pixels wide, and is 40 pixels tall. The figure out what the upper-left corner of an oval is, draw a rectangle around it. The upper-left corner of that rectangle is considered to be the upper-left corner of the oval.



The four numbers in the parentheses are as follows:

The first number is the *x*-coordinate of the upper-left corner of the oval.
The second number is the *y*-coordinate of the upper-left corner of the oval.
The third number is the width, in pixels, of the oval.
The fourth number is the height, in pixels, of the oval.

To create a rectangle:

Use a line of code like the following:

```
GRect ron = new GRect(10, 20, 30, 40);
```

Here, we have created a rectangle named `ron` that has its upper-left corner at the point (10, 20), is 30 pixels wide, and is 40 pixels tall. The four numbers in the parentheses are as follows:

The first number is the *x*-coordinate of the upper-left corner of the rectangle.
The second number is the *y*-coordinate of the upper-left corner of the rectangle.
The third number is the width, in pixels, of the rectangle.
The fourth number is the height, in pixels, of the rectangle.

To create a text message:

Use a line of code like the following:

```
GLabel ginny = new GLabel("I'm Ginny!", 10, 20);
```

Here, we have created a text message named `ginny` that reads "I'm Ginny!". The *lower*-left corner of the message is at the point (10, 20). The items in the parentheses are as follows:

The first item in the parentheses is the content of the message, always enclosed in double-quotes.

The second item is the *x*-coordinate of the *lower*-left corner of the message.

The third item is the *y*-coordinate of the *lower*-left corner of the message.

Once you have created shapes, there are a number of ways of manipulating them.

To make a shape filled-in with a solid color (only for ovals and rectangles):

If you have an oval or rectangle named `dumbledore` that you would like to fill in, use this line:

```
dumbledore.setFilled(true);
```

To make a filled-in shape hollow again (only for ovals and rectangles):

If you would like to make `dumbledore` hollow again, try this:

```
dumbledore.setFilled(false);
```

To change the color of a shape:

If you have a shape named `mcGonagall` that you would like to make a different color, say gray, use this line:

```
mcGonagall.setColor(Color.GRAY);
```

You can also distinguish between the *fill color* and the *pen color* of a shape. The fill color is what's on the inside; the pen color is what's on the outside. There is no command to set the pen color independently of the fill color, but you *can* just set the fill color, like so:

```
mcGonagall.setFillColor(Color.DARK_GRAY);
```

The available colors are the following: `Color.BLACK`, `Color.BLUE`, `Color.CYAN`, `Color.DARK_GRAY`, `Color.GRAY`, `Color.GREEN`, `Color.LIGHT_GRAY`, `Color.MAGENTA`, `Color.ORANGE`, `Color.PINK`, `Color.RED`, `Color.WHITE`, and `Color.YELLOW`. Two colors you may not know are cyan and magenta. Cyan is a mix between blue and green, and magenta is just a little more red than purple.

With any color code, you must have `import java.awt.*;` at the top of your file.

To create a custom color:

Creating a custom color is similar to creating a shape. Use a line that looks like this:

```
Color brown = new Color(100, 50, 0);
```

Then, you can use that color using `setColor`, like this:

```
GRect square = new GRect(90, 90, 20, 20);
square.setFilled(true); // so that it is filled in
square.setColor(brown); // use the color we made
```

The numbers in the parentheses for `new Color(…)` signify the strength of the three different primary colors. All three numbers range from 0 to 255. The first number is for red, the next for green, and the last for blue. So, our `brown` contains half as much green as red and no blue at all. Because we are dealing with creating light, and not reflecting light, white is `new Color(255, 255, 255)` and black is `new Color(0, 0, 0)`.

With any color code, you must have `import java.awt.*;` at the top of your file.

To create a translucent custom color:

If you wish to make a filled-in shape partially see-through, you need to create a color with four numbers, like this:

```
Color blueShade = new Color(0, 0, 255, 127);
```

Like any custom color, you can use `setColor` to give a shape the `blueShade` color. This will make that shape blue and see-through.

The fourth number used when creating the color is called an *alpha* value. It is a number between 0 and 255 that tells how opaque the color is. An alpha of 255 is fully opaque and an alpha of 0 is fully transparent. In other words, an alpha of 255 is the same as not giving an alpha value (colors are, by default, opaque), and an alpha of 0 leads to an invisible shape. Experiment to get a better feel for it.

Like with the other color code, you need to `import java.awt.*;`.

To make a shape disappear:

If you have a shape named `voldemort` that you would like to become invisible, use this line:

```
voldemort.setVisible(false);
```

To make an invisible shape reappear:

To bring `voldemort` back, use this line:

```
voldemort.setVisible(true);
```

To change the coordinates of a shape:

To move a shape named `hagrid` from anywhere to the point (30, 40), use this line of code:

```
hagrid.setLocation(30, 40);
```

If `hagrid` is an oval or rectangle, this moves its upper-left corner to the point (30, 40). If `hagrid` is a line, this moves its starting point to (30, 40); the endpoint follows along, giving the line the same direction and length. If `hagrid` is a message, this puts its lower-left corner at (30, 40).

The first number in the parentheses is the *x*-coordinate of the new point.
The second number in the parentheses is the *y*-coordinate of the new point.

To move a shape a certain distance:

To move a shape named `snape` to the right 5 units and down 20, use this line:

```
snape.move(5, 20);
```

This differs from the `setLocation` command because it moves a shape relative to its starting location. So, if `snape` were at the point (10, 15) and we were to say `snape.move(5, 20);`, `snape` would end up at (15, 35) because 10 + 5 = 15 and 15 + 20 = 35.

This first number in the parentheses is the horizontal distance (positive or negative) the shape will move, and the second number is the vertical distance (positive or negative) the shape will move.

To change the size of a shape (only for ovals and rectangles):

To change the size of a shape named `malfoy` so that its width is 100 and its height is 150, use this line

```
malfoy.setSize(100, 150);
```

The first number is the new width and the second number is the new height. The upper-left corner of the shape remains in the same place.

To scale a shape (not for messages):

If you have a shape, say `myrtle`, of a certain size and you wish to make it twice as big, use this line:

```
myrtle.scale(2);
```

The number in the parentheses is the scale factor. To get the new size of the shape, multiply its old size by this scale factor. The factor can be less than one. For example, `myrtle.scale(.5);` would make `myrtle` half the size she was beforehand.

To move and resize a shape (only for ovals and rectangles):

To completely reset a shape's size and position, you use `setBounds`. For example, if want to take `dementor` and put its upper-left corner at point (50, 100) and make it 25 pixels wide by 25 pixels tall, use this line:

```
dementor.setBounds(50, 100, 25, 25);
```

The four numbers in the parentheses behave exactly as they would if they appears in the parentheses following `new GOval` or `new GRect`.

To move an endpoint of a line:

If you have a line `hogwarts`, and you want to reposition its endpoint at (10, 10), use this line:

```
hogwarts.setEndPoint(10, 10);
```

The first number in the parentheses is the *x*-coordinate of the new endpoint. The second number in the parentheses is the *y*-coordinate of the new endpoint.

There is also `setStartPoint` that works similarly, but it affects the other point.

To change the text in a message:

If you have a message `tomRiddle` and you wish it to read "Voldemort", do this:

```
tomRiddle.setLabel("Voldemort");
```

The item in the parentheses is the new message, enclosed in double-quotes.

To change font size:

If you have a label `alohamora` and you would like to change its font size, use code like this:

```
alohamora.setFont(new Font("Serif", Font.BOLD, 18));
```

The first item in the parentheses is the font name. Choose from the following:

- `"Dialog"`
- `"DialogInput"`
- `"Monospaced"`
- `"Serif"`
- `"SansSerif"`
- `"Default"`

Possibilities for the style of the font (the second parameter to `new Font(...)`) are the following:

- `Font.PLAIN`
- `Font.BOLD`
- `Font.ITALIC`
- `Font.BOLD | Font.ITALIC` (that's a single vertical bar in there)

The third parameter is the size, in points (the usual font measurement), of the font.

To change fonts, you must have `import java.awt.*;` at the top of your file.

To change the order of shapes:

If you have a shape `filch` that's lurking near the back (in other words, covered up by other filled-in shapes) that you wish to move to the front, use this line:

```
filch.sendToFront();
```

This will put `filch` in front of (covering) all the other shapes. There exists also this line:

```
filch.sendToBack();
```

This would put `filch` behind all the other shapes.