Learning Activity: Graphics

Back

Objectives

- Gain familiarity with creating graphical components.
- Gain familiarity with using java.awt.Graphics.
- Gain familiarity with the standard syntax for properties.

Instructions

All of these exercises for the learning activity are based on the X class, downloadable here, X.java. First we will gain some familiarity with updating drawing code. Here is the initial version of X.java, sans comments.

```
public class X extends javax.swing.JComponent {
    public X() {
        setPreferredSize(new java.awt.Dimension(25, 25));
    }
    public void paintComponent(java.awt.Graphics g) {
        super.paintComponent(g);
        g.drawLine(0, 0, getWidth(), getHeight());
        g.drawLine(0, getHeight(), getWidth(), 0);
    }
}
```

You can see what this component looks like by adding it to a window.

```
javax.swing.JFrame win;
win = new javax.swing.JFrame("Test Component");
X x = new X();
win.add(x);
win.setSize(400, 300);
win.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
win.setVisible(true);
```

You will notice that this doesn't display quite right. The ends of the 'X' are cut off. This is even more apparent if you put them into the edges of the BorderLayout. (Recall this is the default layout for the content pane of a JFrame.) (Notice also that the effects of setting the preferred size in the constructor are now apparent, as well.)

```
javax.swing.JFrame win;
win = new javax.swing.JFrame("Test Component");
win.add(new X(), java.awt.BorderLayout.NORTH);
win.add(new X(), java.awt.BorderLayout.SOUTH);
win.add(new X(), java.awt.BorderLayout.EAST);
win.add(new X(), java.awt.BorderLayout.WEST);
win.setSize(400, 300);
win.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
win.setVisible(true);
```

Here is a video of this application.

In class, we fixed this by adjusting the endpoints for the two lines. Remember that drawing lines uses the pen and that the pen fills the pixel which is down and to the right of the coordinate (an infinitesimally small point).

```
public void paintComponent(java.awt.Graphics g) {
  super.paintComponent(g);
  g.drawLine(0, 0, getWidth() - 1, getHeight() - 1);
  g.drawLine(0, getHeight() - 1, getWidth() - 1, 0);
}
```

The subclass can have new fields and methods. In class, we added support for a lineColor property, with a private field and get and set methods.

```
private java.awt.Color lc;
public java.awt.Color getLineColor() {
  return lc;
}
public void setLineColor(java.awt.Color rgb) {
  lc = rgb;
  repaint();
}
```

Notice this follows the classic pattern for getter and setting methods for a property. The getter takes no parameters but returns the type for the property. The setter, on the other hand, takes a parameter of the property type, but returns nothing. Notice that the getter and the setter in the simplest case would be equivalent to making the field public. However, that is not the case here, we have added a repaint call to the setter, so whenever the lineColor property is set, the component will request a repaint. Also, now that we have added a field to the class, it makes sense to have the constructor set a value for the field. By default, the lineColor shall be set to black.

Notice also that this property consists of two methods, a getter and a setter. Even though as we work through this learning activity, we will only use the setLineColor method, the getLineColor method was included for "completeness". This is a minimal step that can help make the new component potentially reusable.

Obviously, the rendering code in paintComponent must be updated to use the new property.

```
public void paintComponent(java.awt.Graphics g) {
  super.paintComponent(g);
  g.setColor(lc);
  g.drawLine(0, 0, getWidth() - 1, getHeight() - 1);
  g.drawLine(0, getHeight() - 1, getWidth() - 1, 0);
}
```

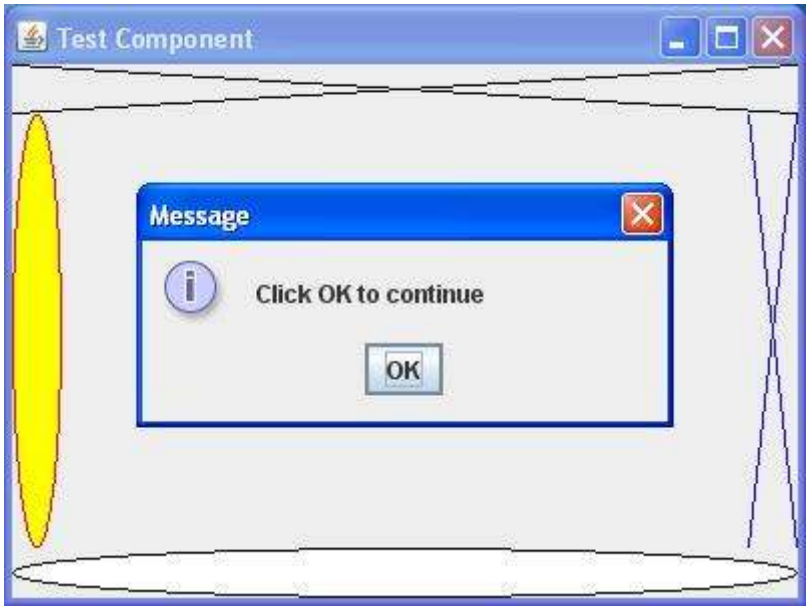
We're just about done. Now, the field will have the value null by default. The Graphics g will not be very happy setting its color to null. So, we updated the constructor to set an initial value for the field. java.awt.Color.black is a reasonable initial value for the line color.

We have made enough changes to the **X** class that we will rename it, **X2**.

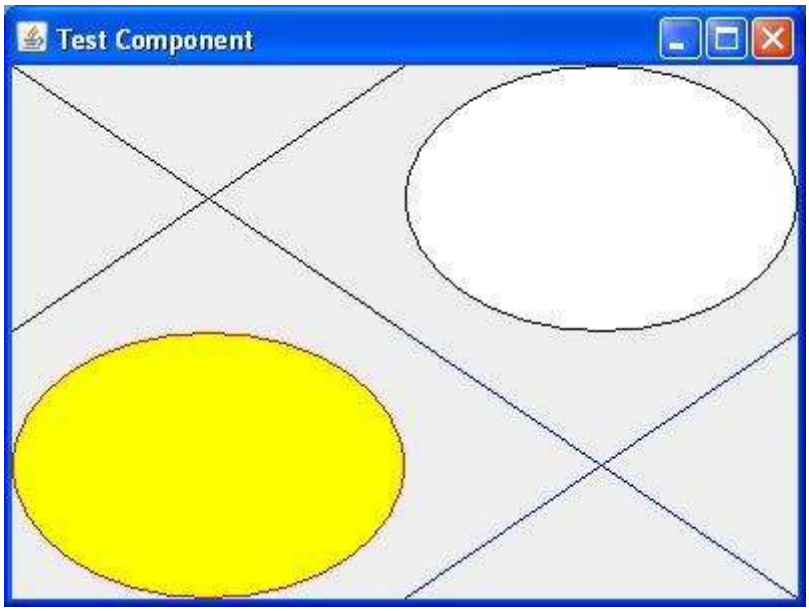
Your task for this lab is to create a 'O' component to go along with the updated 'X' component, **X2**. Name the new component **O2**, to match the name X2. The O2 class should render itself completely within the component, with the ellipse touching each of the edges of the component. This O2 component should have two novel (new) properties, lineColor and fillColor. Both of these should be of type java.awt.Color. The default lineColor shall be black; the default fillColor, white. Make sure you document your new class.

Use this class to test the components: GraphicsLA.java.

This should first display this:



Then it should display this:



Grading Rubric

Check for X2 and O2 that generate the expected output and that have reasonable style.

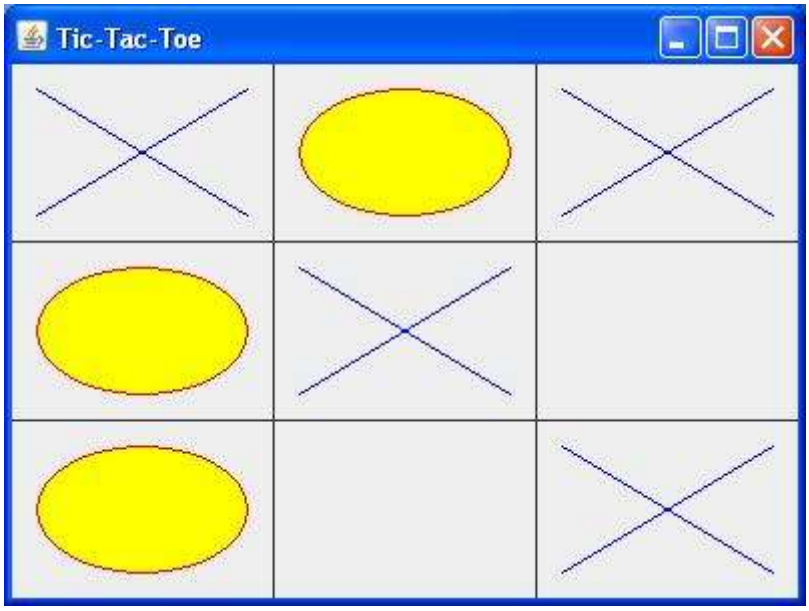
Minus if X2 or O2 do not perform to specification, or if the style is not up to par.

Style Guidelines

- The classes shall have comment headers which include your name and this lab designation (Lab 2: Graphics).
- Each method shall have a comment which precedes it that briefly describes the function of the method.
- The properties shall have both get and set methods.
- The code between curly braces shall be indented 2 or more spaces than the left margin of the braces.
- Code at the same level of indentation shall align vertically.

Plus Option

Create a tic-tac-toe board using your 'X' and 'O' components as markers. The target for the output is this:



Note that the X2 and O2 components have ***NOT*** been altered for this part of the learning activity, they still draw to the edges of the component. To create this image, I only used layout managers we have seen: BorderLayout, GridLayout. To draw the lines, I did need to use inheritance and override paintComponent. The simplest solution I have found also creates a third component type, a blank one.

To submit the plus option, name the class **TTT**. The TTT is for tic-tac-toe. The main method in TTT should instantiate the tic-tac-toe board object and then add the four X2 instances and three O2 instances, not necessarily in that order. You will probably need to also add two or more instances of a "placeholder" graphics object. Any additional classes, like the "placeholder" class or a class for the tic-tac-toe board shall be package private classes in the TTT.java file.

Submission

For the Minus or Check:

- X2.java
- O2.java

For the Plus:

- X2.java
- O2.java
- TTT.java (any additional helper classes are to be package private in TTT.java)

Back