1. Here is a simple event driven program. What is missing from it? That is, what needs to be added to this program so that it compiles, runs, and noticeably responds to events? (Note: Do not delete any code from the program. Only add code to it.)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class WhatIsMissing
{
   public WhatIsMissing() {
      final JFrame jf = new JFrame("WhatIsMissing");
      jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      jf.setLayout(new FlowLayout());
      final JButton jb = new JButton("Hello");
      jb.addActionListener(this);
      jf.add(jb);
      jf.setLocationRelativeTo(null);
      jf.setSize(400, 400);
   }

   @Override public void mouseClicked(MouseEvent e) {
      System.out.println(e);
   }
   @Override public void componentMoved(ComponentEvent e) {
      System.out.println(e);
   }
   @Override public void componentResized(ComponentEvent e) {
      System.out.println(e);
   }

}
```

2. Explain how this program works. You should be able to explain the purpose of every line of code.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Review_Problem_2 implements ActionListener {
   private boolean checked = false;
   public Review_Problem_2() {
      final JFrame jf = new JFrame("Review Problem 2");
      jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      jf.setLayout(new FlowLayout());
      final JButton   jb1 = new JButton("Hello");
      final JButton   jb2 = new JButton("GoodBy");
      final JCheckBox cb1 = new JCheckBox("Confuse Them");
      jf.add(jb1);  jf.add(jb2);  jf.add(cb1);
      jf.setLocationRelativeTo(null);
      jf.setSize(400, 400);
      jf.setVisible(true);
      cb1.addActionListener(this);
      jb1.addActionListener(new ActionHandler());
      jb2.addActionListener(new ActionListener(){
         @Override public void actionPerformed(ActionEvent e) {
            System.out.println(!checked ? "Goodbye" : "Hello?");
      }});
   }
   @Override public void actionPerformed(ActionEvent e) {
      checked = ! checked;
   }
   class ActionHandler implements ActionListener {
      @Override public void actionPerformed(ActionEvent e) {
         System.out.println(!checked ? "Hello" : "Goodbye?");
      }
   }
   public static void main(String[] args) {
      new Review_Problem_2();
   }
}
```

3. The following block of code creates a `FrameBuffer` with two viewports, places a model of a disk (with radius 1) at the center of the image plane, then it sets up a view-rectangle for each viewport and renders the view-rectangle into its viewport. Draw a sketch of the contents of the `FrameBuffer` after this code executes. In your sketch, label each of the two `Viewports`, `vp1` and `vp2`. See the last page of this document about the view-rectangle and viewport APIs.

```
FrameBuffer fb = new FrameBuffer(400, 200, Color.gray);
FrameBuffer.Viewport vp1 = fb.new Viewport(0, 0, 200, 100, Color.white);
FrameBuffer.Viewport vp2 = fb.new Viewport(200, 100, 200, 100, Color.white);
vp1.clearVP();
vp2.clearVP();
Scene scene = new Scene();
Model disk = new Disk(1.0, 1, 64);
ModelShading.setColor(disk, Color.black);
scene.addPosition( new Position(disk) );
scene = scene.changeCamera(Camera.projOrtho(-1.0, 1.0, 0.0, 1.0);
Pipeline.render(scene, vp1);
scene = scene.changeCamera(Camera.projOrtho(-1.0, 1.0, -1.0, 0.0);
Pipeline.render(scene, vp2);
```

4. The following block of code creates a framebuffer, places a model of a disk (with radius 1) at the center of the image plane, then it sets up the view-rectangle and the viewport, and finally it renders the view-rectangle into the viewport. In each part below, give a schetch of the view-rectangle and the viewport. See the last page of this document about the view-rectangle and viewport APIs.

```
final FrameBuffer fb = new FrameBuffer(500, 500, Color.gray);
Scene scene = new Scene();
final Model disk = new Disk(1.0, 1, 64);
ModelShading.setColor(disk, Color.black);
scene.addPosition( new Position(disk) );
/**** Declarations of: left, right, bottom, top, x, y, w, h ****/
// Set up the view-rectangle (in the image plane).
scene = scene.changeCamera(Camera.projOrtho(left, right, bottom, top));
// Set up the viewport (in the framebuffer).
FrameBuffer.Viewport vp = fb.new Viewport(x, y, w, h, Color.white);
vp.clearVP();
Pipeline.render(scene, vp);
```

(a)
```
// Parameterize the view-rectangle.
double left   = -0.5,  right = 1.0;
double bottom = -0.5,  top   = 1.0;
// Parameterize the viewport.
int x = 50,   y = 50;
int w = 400,  h = 400;
```

(b)
```
// Parameterize the view-rectangle.
double left   = -1.0,  right = 1.0;
double bottom =  0.0,  top   = 1.0;
// Parameterize the viewport.
int x = 0,    y = 250;
int w = 500,  h = 250;
```

(c)
```
// Parameterize the view-rectangle.
double left   = 0.0,  right = 1.0;
double bottom = 0.0,  top   = 1.0;
// Parameterize the viewport.
int x = 250,  y = 0;
int w = 250,  h = 500;
```

(d)
```
// Parameterize the view-rectangle.
double left   = -1.0,  right = 3.0;
double bottom = -1.0,  top   = 3.0;
// Parameterize the viewport.
int x = 0,    y = 0;
int w = 400,  h = 400;
```

5. Give one important reason for using homogeneous coordinates in a graphics system.

6. Demonstrate, using algebra or pictures, that order matters when composing (concatenating) two transformations.

7. (a) Compute the 16 number that make up the 4×4 homogeneous matrix in the `Position` object after executing the following lines of code.

```
position.transform( Matrix.translate(3.0, 0.0, 0.0)
            .times( Matrix.scale(3.0, 1.0, 1.0) ) );
```

   (b) Compute the 16 number that make up the 4×4 homogeneous matrix in the `Position` object after executing the following lines of code.

```
position.transform( Matrix.scale(3.0, 1.0, 1.0)
            .times( Matrix.translate(3.0, 0.0, 0.0) ) );
```

   (c) Compute the 16 number that make up the 4×4 homogeneous matrix in the `Position` object after executing the following lines of code.

```
position.transform( Matrix.scale(3.0, 1.0, 1.0)
            .times( Matrix.translate(1.0, 0.0, 0.0) ) );
```

   (d) How would you explain to someone, without doing the actual calculations, why the matrices in parts (a) and (b) are different, but the matrices in parts (a) and (c) are the same?

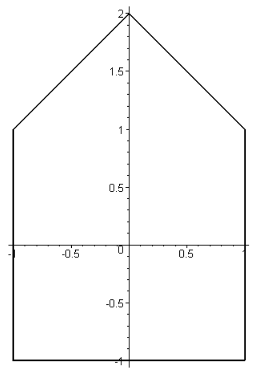
8. Assume that `p` is a reference to a Position object and that `m1`, `m2`, `m3` are references to `Matrix` objects. In what ways are the following two lines of code similar, and in what ways do they differ?

```
p.transform( m1.times(m2).times(m3) );
p.transform( m1.times(m2.times(m3)) );
```

9. Assume that `p` is a reference to a `Position` object. What is wrong with the following line of code? Explain in detail what the line of code does. How might you fix it?

```
p.getMatrix().times(Matrix.translate(3,0,0).times(Matrix.scale(2,2,2)));
```

10. (a) Suppose we have defined a subclass `DrawModel` of `Model` that represents the following image in the xy-plane. Draw a picture of the scene graph that the following code constructs. Draw a sketch of the scene that the code would produce in the camera's image plane when the scene is rendered. For each of the three model images, give the coordinates of the model's origin point.

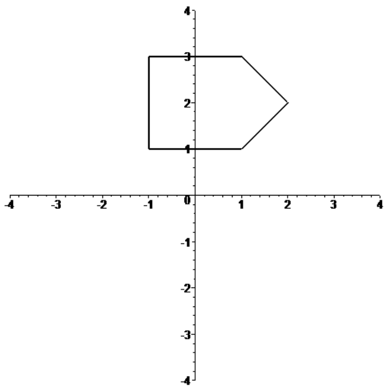


```
Model drawModel = new DrawModel();
Position p1 = new Position(drawModel);
Position p2 = new Position(drawModel);
Position p3 = new Position(drawModel);
p2.transform( Matrix.translate(1.0, 3.0, 0.0).times(
              Matrix.rotateZ(45.0)) );
p3.transform( Matrix.translate(1.0, 0.0, 0.0).times(
              Matrix.translate(1.0, 0.0, 0.0)) );
p1.transform( Matrix.translate(-2.0, -2.0, 0.0).times(
              Matrix.rotateZ(-90.0)) );
Scene scene = new Scene();
scene.addPosition(p1, p2, p3);
```

(b) With the addition of the following code, what is drawn in the framebuffer?

```
ModelShading.setColor(drawModel, Color.black);
scene = scene.changeCamera(Camera.projOrtho(-2.0, 2.0, -2.0, 3.0));
FrameBuffer fb = new FrameBuffer(500, 600, Color.gray);
FrameBuffer.Viewport vp = fb.new Viewport(50,50,400,500,Color.white);
vp.clearVP();
Pipeline.render(scene, vp);
```

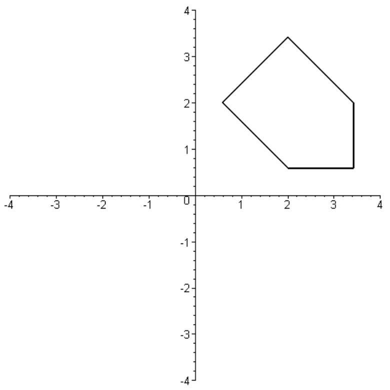
11. Using the same `DrawModel` class from the previous problem, write code that would generate the following image. Do this two different ways, one way with a translation preceding a rotation (in the code), and the second way with a rotation preceding a translation (in the code).



(a) Translation and then rotation.

(b) Rotation and then translation.

12. Using the same `DrawModel` class from the previous problem, write code that would generate the following image. Do this two different ways, one way with a translation preceding a rotation (in the code), and the second way with a rotation preceding a translation (in the code).



(a) Translation and then rotation.

(b) Rotation and then translation.

13. Suppose you have available to you a subclass `DrawUnitCircle` of `Model` that draws in the xy-plane a circle of radius one centered at the origin. Suppose that `theta1` and `theta2` are each defined as a (different) non-zero double. What would the following code draw in the camera's image plane? What does its scene graph look like?

```
Model circle = new DrawUnitCircle();
Position p1 = new Position(circle);
Position p2 = new Position(circle);
Position p3 = new Position(circle);
p1.transform( Matrix.scale(2.0, 2.0, 2.0) );
p2.transform( Matrix.rotateZ(theta1).times(
              Matrix.translate(3.0, 0.0, 0.0)) );
p3.transform( Matrix.rotateZ(theta2).times(
              Matrix.translate(1.5, 0.0, 0.0)).times(
              Matrix.scale(0.5, 0.5, 1.0)) );
Scene scene = new Scene();
scene.addPosition(p1, p2, p3);
```

14. Define a `Matrix` object `m` and a `Position` object `p`. What does this line of code do?

```
p.transform(p.getMatrix().times(m));
```

How does it differ from this line of code?

```
p.transform(m.times(p.getMatrix()));
```

15. Define `Matrix` objects `m1`, `m2`, `m3`, `m4` and a `Position` object `p`. Write a formula for the final `Matrix` in `p`.

```
p.transform(m1);
p.transform(m2);
p.transform( m3.times(p.getMatrix()) );
p.transform( p.getMatrix().times(p.getMatrix()) );
p.transform( m4.times( p.getMatrix() ) );
```

16. Define `Matrix` objects `m1`, `m2`, `m3` and a `Position` object `p`. Simplify this code as much as possible.

```
p.transform(Matrix.identity());
p.transform(p.getMatrix().times(m1));
p.transform(m2.times(p.getMatrix()));
p.transform(p.getMatrix().times(m3));
```

17. On the second to last page of this document there is a definition of a simple, 2-by-2, matrix class. Use that class definition to answer these questions.

    Given three `Mat2` objects `m1`, `m2`, `m3`, in each part below explain in detail what the line of code does. How many `Mat2` objects are instantiated? Which ones are mutated and which ones are garbage collected? In what order are the matrix calculations done?

    (a) `Mat2 m4 = m1.mult(m2).times(m3);`

    (b) `Mat2 m5 = m1.mult(m2.times(m3));`

    (c) `Mat2 m6 = m1.times(m2).mult(m3);`

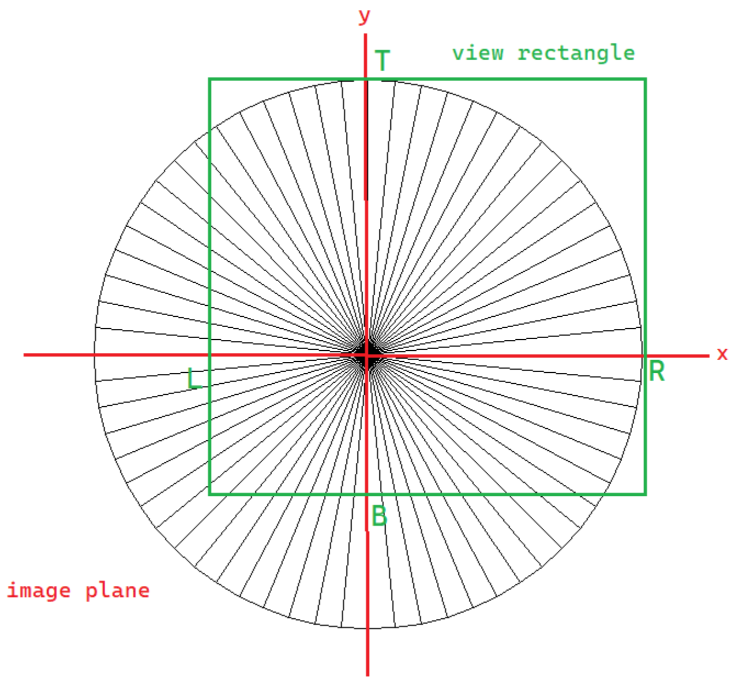
    (d) `Mat2 m7 = m1.times(m2.mult(m3));`

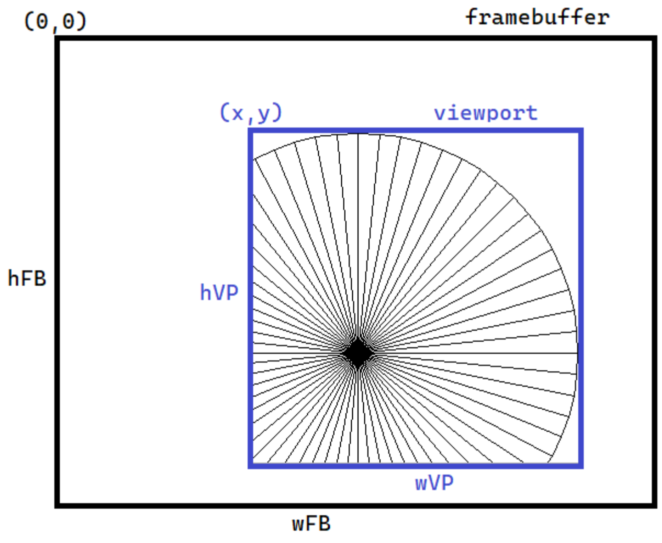
    (e) `Mat2 m8 = new Mat2(5,0,0,5).mult(m1).mult( m2.times(m3) );`

    (f) How does the next line of code compare to the previous line?

    ```
    Mat2 m9 = new Mat2(5,0,0,5).mult(m1.mult( m2.times(m3) ));
    ```

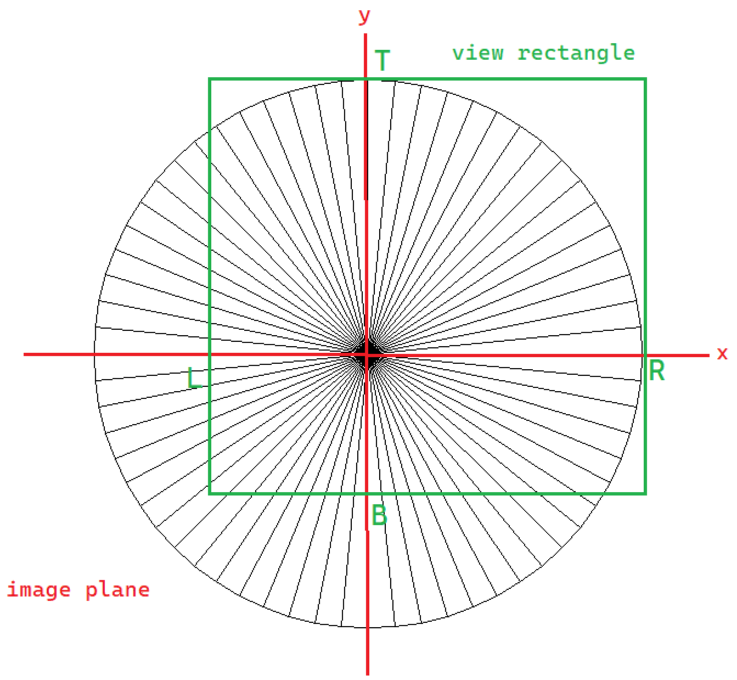
```
class Mat2
{
   private double m00, m01, m10, m11;

   public Mat2(double m00, double m01, double m10, double m11)
   {
      this.m00 = m00;
      this.m01 = m01;
      this.m10 = m10;
      this.m11 = m11;
   }

   public Mat2 times(final Mat2 that)
   {
      final double a00 = this.m00 * that.m00 + this.m01 * that.m10;
      final double a01 = this.m00 * that.m01 + this.m01 * that.m11;
      final double a10 = this.m10 * that.m00 + this.m11 * that.m10;
      final double a11 = this.m10 * that.m10 + this.m11 * that.m11;
      return new Mat2(a00, a01, a10, a11); // Return a new matrix.
   }

   public Mat2 mult(final Mat2 that)
   {
      final double a00 = this.m00 * that.m00 + this.m01 * that.m10;
      final double a01 = this.m00 * that.m01 + this.m01 * that.m11;
      final double a10 = this.m10 * that.m00 + this.m11 * that.m10;
      final double a11 = this.m10 * that.m10 + this.m11 * that.m11;
      this.m00 = a00; // Mutate this Mat2 object.
      this.m01 = a01;
      this.m10 = a10;
      this.m11 = a11;
      return this; // For method chaining.
   }

   public String toString(){return "["+m00+",\t"+m01+"]\n["+m10+",\t"+m11+"]";}
}
```

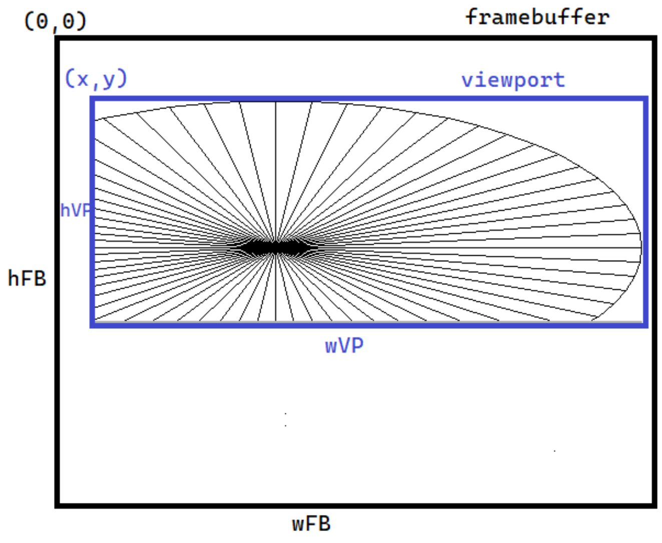




```
Camera.projOrtho(left, right, bottom, top)
 or
Camera.projPerspective(left, right, bottom, top)
```

```
FrameBuffer fb = new FrameBuffer(wFB, hFB)
FrameBuffer.Viewport vp = fb.new Viewport(x, y, wVP, hVP)
```





```
Camera.projOrtho(left, right, bottom, top)
 or
Camera.projPerspective(left, right, bottom, top)
```

```
FrameBuffer fb = new FrameBuffer(wFB, hFB)
FrameBuffer.Viewport vp = fb.new Viewport(x, y, wVP, hVP)
```