1. What is a Framebuffer data structure? What does it contain? What does it represent? How is it used in a graphics rendering pipeline?
2. What is a Scene data structure? What does it contain? What does it represent? How is it used in a graphics rendering pipeline?
3. (a) Briefly describe the contents of an `obj` file.
   (b) What role do `obj` files play in a graphics system?
   (c) How can a `obj` file parser make use of the `LineLoop` primitive?
4. (a) Briefly describe the contents of an `grs` file.
   (b) How can a `grs` file parser make use of the `LineStrip` primitive?
5. Briefly describe the contents of a ppm file. As an image file format, what advantage does it have? What is its disadvantage?
6. Briefly describe each of the following coordinate systems.
   (a) model coordinates
   (b) world coordinates
   (c) view coordinates
7. Put the following coordinate systems into the proper order, starting with the beginning of the graphics rendering pipeline and moving to the framebuffer.
   - view coordinates
   - normalized device coordinates
   - model coordinates
   - clip coordinates
   - world coordinates
   - viewport coordinates
8. Briefly describe each of the following stages of the graphics rendering pipeline.
   (a) Model2World
   (b) World2View
   (c) BackFaceCulling
   (d) Projection (also called View2Clip)
   (e) Clip2Viewport
   (f) Rasterize

9. (a) Version 5 of our renderer has five types of geometric primitives (subclasses of Primitive.java). Give the name and a brief description of these five geometric primitives.

   (b) Briefly explain why we added these primitives to the renderer.

10. When is it preferable to use an orthographic viewing projection? When is it preferable to use perspective projection? Explain why.

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

12. Demonstrate, using algebra or pictures, that order matters when composing two transformations.

13. What would be the 16 entries in the $4 \times 4$ homogeneous matrix in the `Model` object after executing the following lines of code?

```
model.model2Identity();
model.modelTranslate(1.5, 2.0, -3.0);
model.modelScale(3, 2, 0.5);
```

14. What would be the 16 entries in the $4 \times 4$ homogeneous matrix in the `Model` object after executing the following lines of code?

```
model.model2Identity();
model.modelRotate(90, 0, 0, 1);
```

15. (a) Compute the 16 number that make up the $4 \times 4$ homogeneous matrix in the `Model` object after executing the follow lines of code.

```
model.model2Identity();
model.modelTranslate(3.0, 0.0, 0.0);
model.modelScale(3.0, 1.0, 1.0);
```

   (b) Compute the 16 number that make up the $4 \times 4$ homogeneous matrix in the `Model` object after executing the follow lines of code.

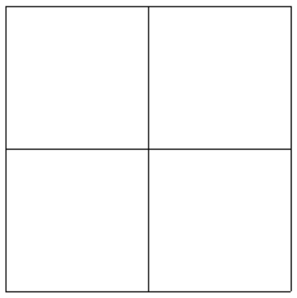
```
model.model2Identity();
model.modelScale(3.0, 1.0, 1.0);
model.modelTranslate(3.0, 0.0, 0.0);
```

   (c) Compute the 16 number that make up the $4 \times 4$ homogeneous matrix in the `Model` object after executing the follow lines of code.

```
model.model2Identity();
model.modelScale(3.0, 1.0, 1.0);
model.modelTranslate(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?

16. Use appropriate primitives to define a model for the following figure in such a way that you send at most eight `Vertex` objects into the graphics pipeline. If you send the same vertex object twice into the pipeline, that counts as two objects. You can assume that this figure is centered at the origin and the sides have length two.



17. Consider the following block of code.

```
Vertex[] v = {new Vertex( 0,  0, 0),
              new Vertex(-1,  1, 0),
              new Vertex( 1,  1, 0),
              new Vertex( 2,  0, 0),
              new Vertex( 1, -1, 0),
              new Vertex(-1, -1, 0)};
Primitive p1 = new LineFan(v);
Primitive p2 = new LineLoop(v);
Model m = new Model();
m.addPrimitive(p1);
m.addPrimitive(p2);
```

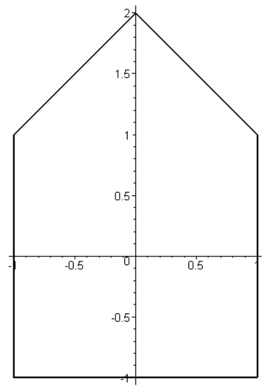
(a) What picture would be drawn if we rendered the model `m` built by this code (assume an orthographic projection looking down the $z$-axis)? In your picture, label the appropriate points in the order v0 to v5.

(b) How many Java objects does this code instantiate (count the objects that are composed in objects like `Model` and `Primitive`)? Draw a detailed picture of what the objects "look like" in the Java heap (which objects refer to which objects?).

(c) Right after the primitive assembly stage of the rendering pipeline, how many `LineSgement` objects and how many `Vertex` objects would the model `m` refer to?

(d) Write a minimal(!) Java program that will compile and run and draw the above model (and by "draw" I mean write the `FrameBuffer` to a file that can be viewed).

18. Suppose we have defined a subclass `DrawModel` of `Model` that represents the following image in the $xy$-plane. Draw a sketch of the scene that the following code would produce. For each of the three model images, give the coordinates of the model's "origin" point.



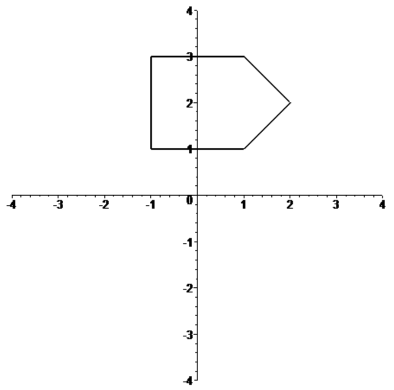
```
Model m1 = new Model();
Model m2 = new Model();
Model m3 = new Model();
Model m4 = new Model();
Model m5 = new Model();
Model drawModel = new DrawModel();

m1.addSubModel(m2);
m1.addSubModel(m3);
m3.addSubModel(m4);
m3.addSubModel(m5);
m2.addSubModel(drawModel);
m4.addSubModel(drawModel);
m5.addSubModel(drawModel);

m3.modelTranslate(1.0, 0.0, 0.0);
m4.modelTranslate(0.0, 3.0, 0.0);
m4.modelRotate(45.0, 0.0, 0.0, 1.0);
m5.modelTranslate(1.0, 0.0, 0.0);
m2.modelTranslate(-2.0, -2.0, 0.0);
m2.modelRotate(-90.0, 0.0, 0.0, 1.0);

Scene scene = new Scene();
scene.addModel(m1);
```

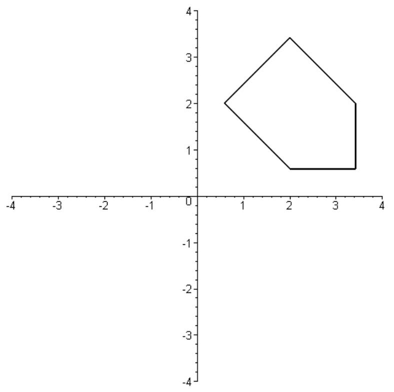
19. Using the same `DrawModel` class from the previous problem, write code that would generate the following image. Do this two different ways, one way with a translation preceding a rotation, and the second way with a rotation preceding a translation.



(a) Translation and then rotation.

(b) Rotation and then translation.

20. 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, and the second way with a rotation preceding a translation.



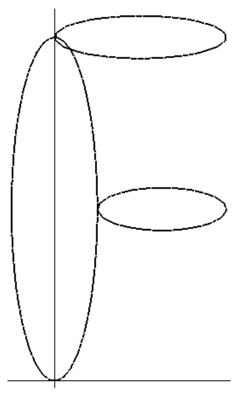
(a) Translation and then rotation.

(b) Rotation and then translation.

21. 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. What would the following code draw in the $xy$-plane?

```
Model m1 = new Model();
Model m2 = new Model();
Model m3 = new Model();
Model m4 = new Model();
Model circle = new DrawUnitCircle();
m1.addSubModel(m2);
m1.addSubModel(m3);
m1.addSubModel(m4);
m2.addSubModel(circle);
m3.addSubModel(circle);
m4.addSubModel(circle);
m2.modelScale(2.0, 2.0, 2.0);
m3.modelRotate(theta1, 0, 0, 1);
m3.modelTranslate(3.0, 0.0, 0.0);
m4.modelRotate(theta2, 0, 0, 1);
m4.modelTranslate(1.5, 0.0, 0.0);
m4.modelScale(0.5, 0.5, 1.0);
```

22. 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. Use a *single* instance of this model, several instances of `Model`, and any of the methods `addSubModel()`, `modelTranslate()`, `modelRotate()`, and `modelScale()`, to draw the following "F-shape". The vertical ellipse has its bottom at the point $(0, 0)$ and its top at the point $(0, 4)$, and at its widest it is 1 unit across. The upper horizontal ellipse is 2 units long. The lower horizontal ellipse is 1.5 units long. Its right hand end point has coordinates $(2, 2)$. The two horizontal ellipses are 0.5 units tall.



23. What does the following code draw in the $xy$-plane? Hint: Draw a picture of the graph data structure built by this code. Then determine what the transformations do to the geometry.

```
Vertex[] v = {new Vertex(-1, 0, 0), new Vertex(1, 0, 0)};
LineSegment ls = new LineSegment(v);
Model m1 = new Model();
m1.addPrimitive(ls);
Model m2 = new Model();
Model m3 = new Model();
Model m4 = new Model();
m2.addSubModel(m3);
m2.addSubModel(m4);
m3.addSubModel(m1);
m4.addSubModel(m1);
m3.modelTranslate(1, 0, 0);
m4.modelTranslate(2, 0, 0);
m4.modelRotate(90, 0,0,1);
m4.modelScale(0.5, 1, 1);
Model m5 = new Model();
Model m6 = new Model();
Model m7 = new Model();
Model m8 = new Model();
m5.addSubModel(m6);
m5.addSubModel(m7);
m5.addSubModel(m8);
m6.addSubModel(m2);
m7.addSubModel(m2);
m8.addSubModel(m2);
m7.modelRotate(90, 0,0,1);
m8.modelRotate(-135, 0,0,1);
```