Week 2 Tutorial Solutions

Question 1:

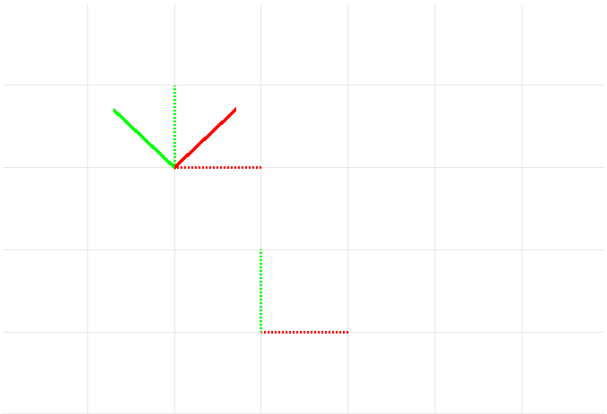
Suppose we compute a coordinate frame with the following sequence of 2d transformations. Draw the successive coordinate frames:

Original Coordinate Frame

```
.translate(-1,2,0)
```

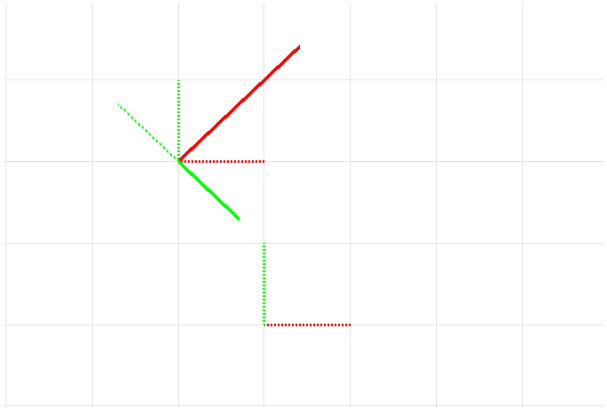
Translated Coordinate Frame (previous frames shown with dotted lines)

```
.rotate(45)
```



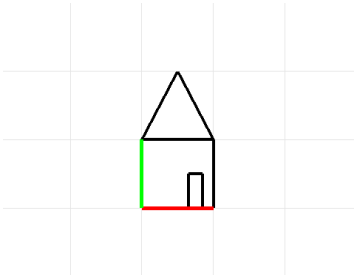
Translated,Rotated Frame (previous frames shown with dotted lines)

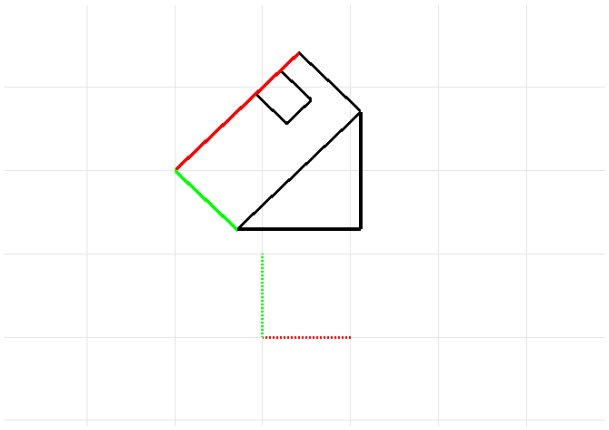
```
.scale(2,-1)
```



Final Translated,Rotated,Scaled Coordinate Frame (previous frames shown with dotted lines)

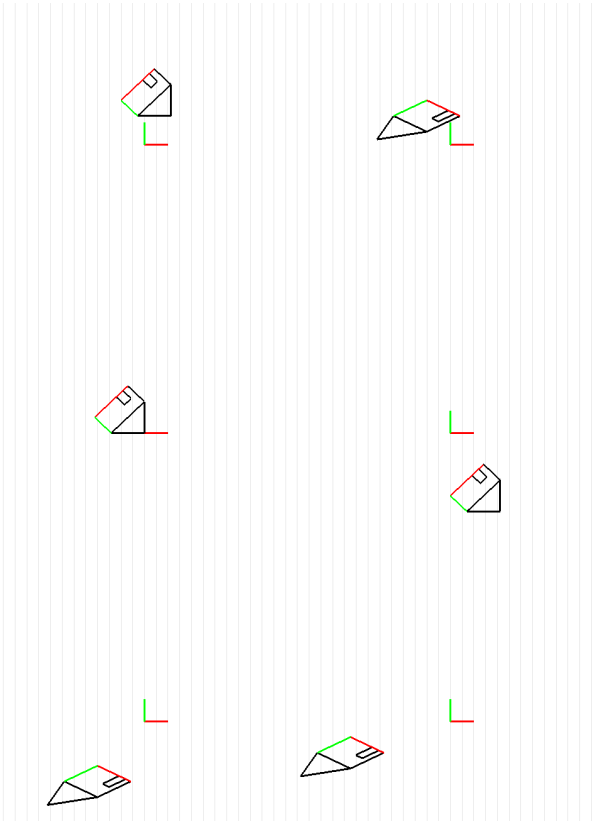
Draw a house in the resulting coordinate frame, assuming that the house was located in the following starting position.





Final House Translated,Rotated,Scaled

How does the result change if you change the order of the transformations (try all 6 possiblities).



Here the order TRS, TSR on the first row, RTS, RST on the second row and STR, SRT on the last row.

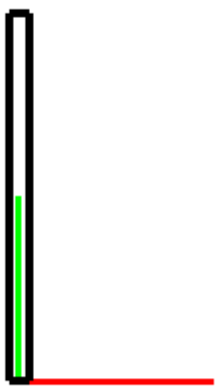
Things to note:

- For all the sequences of transformations that involved scaling before rotation we ended up with axes that were not perpedicular so the image appears skewed. This happens when non-unform scaling occurs before a rotation.
- Rotating before a translation instead of translating before rotating, does not change the axes but changes the origin.

Question 2:

Given a function

```
//This function draws a pole of width 0.1, height 2
//With the bottom, middle of the pole at the origin
//of the given coordinate frame as shown in the following picture.
private void drawPole(GL3 gl, CoordFrame2D frame);
```



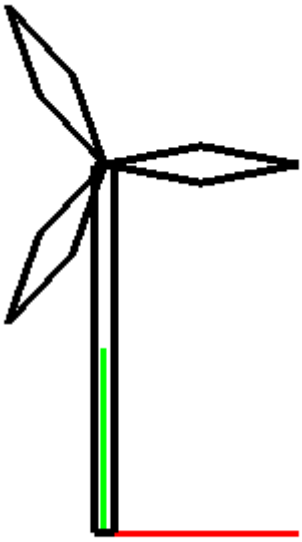
And another function

```
//This function draws a windmill vane with the left most point at the
//origin of the given coordinate frame as shown in the following picture
private void drawVane(GL3 gl, CoordFrame2D frame);
```

a. Write a function called

```
public void drawWindmill(GL3 gl, CoordFrame2D frame);
```

that can draw a windmill as shown in the following picture



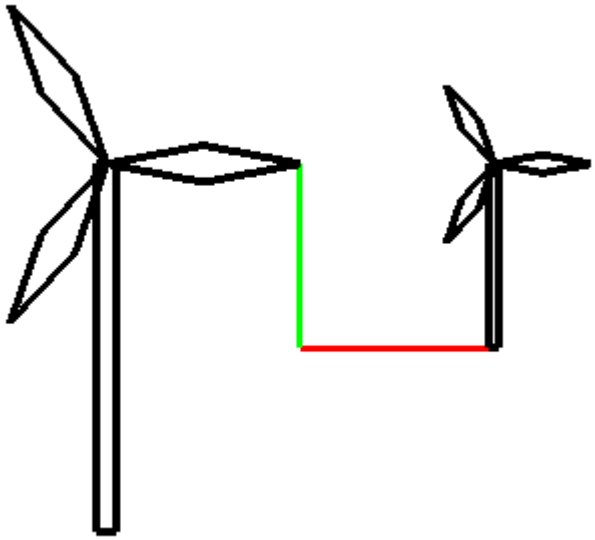
```
public void drawWindmill(GL3 gl, CoordFrame2D frame){
    drawPole(gl, frame);

    frame = frame.translate(0,2);
    for(int i=0; i < 3; i++){
        frame = frame.rotate(120); //These rotations accumulate
        drawVane(gl, frame);
    }
}
```

b. Write a function named

```
public void drawWindmills(GL3 gl, CoordFrame2D frame);
```

that draws windwills as shown in the following picture. Note: This picture shows the world coordinate frame.



```
public void drawWindmills(GL3 gl, CoordFrame2D frame){
    drawWindmill(gl, frame.translate(-1,-1));
    drawWindmill(gl, frame.translate(1,0).scale(0.5,0.5));
}
```

c. Draw a scene tree that could model this simple scene.

```
          Scene Root
          /         \
   Windmill         Windmill
      |                |
     Pole             Pole
    / |  \           / |  \
Vane Vane Vane   Vane Vane Vane
```

d. How could we make our vanes rotate?

Have some kind of variable such as theta, that gets updated each time display gets called by the animator. Then use it to perform a rotation before the vanes are drawn. For example:

```
public void drawWindmill(GL3 gl, CoordFrame2D frame){
    drawPole(gl, frame);

    frame = frame.translate(0,2)
        .rotate(theta)
    for(int i=0; i < 3; i++){
        frame = frame.rotate(120); //These rotations accumulate
        drawVane(gl, frame);
    }
}
```

Question 3:

Consider building a model of the solar system. What would the scene tree look like? What is the local coordinate frame for each object? How would they evolve over time? Careful, this is not as obvious at it might look.

There are many possible answers. A naive tree might look like:

```
Sun --- Mercury
    \-- Venus
    \-- Earth -- Moon
```

```
    \-- Mars
     ... etc
```

The problem with this model is that the Moon is directly attached to the Earth's coordinate frame. So when the Earth rotates, the Moon will move with it.

A better tree would be

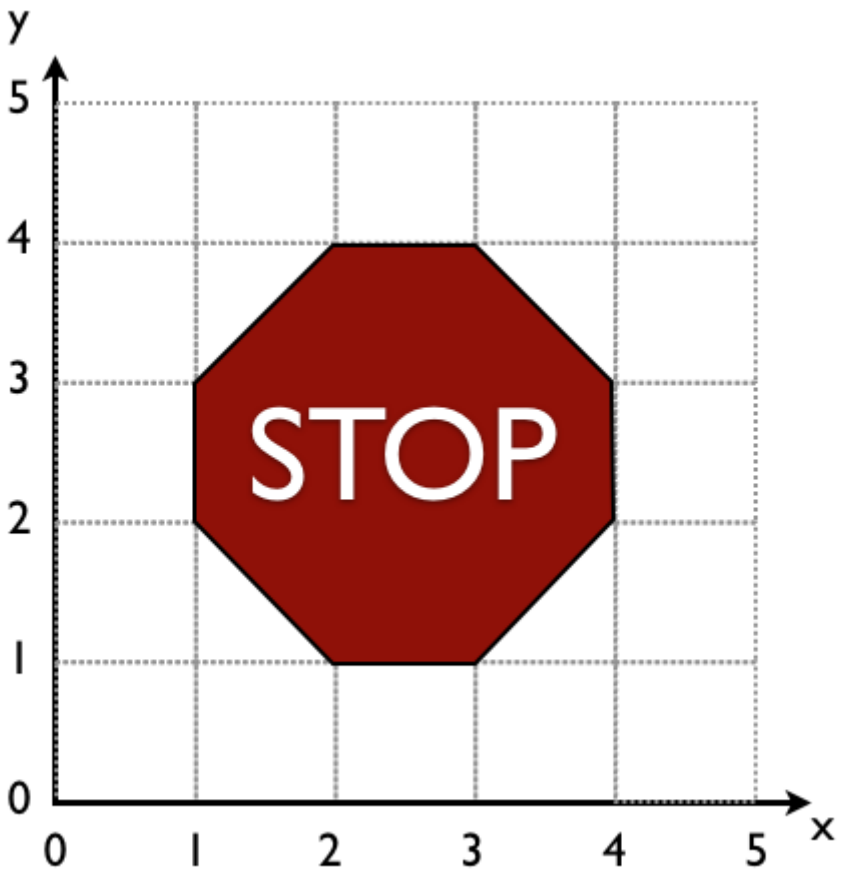
```
Sun --- Mercury
    \-- Venus
    \-- EarthPivot --- Earth
    |              \-- Moon
    \-- Mars
     ... etc
```

The EarthPivot is an empty object, just a place holder for a coordinate frame. The EarthPivot orbits the sun. The Earth sits at the EarthPivot and rotates. The Moon orbits around the EarthPivot. Now the Moon's orbit is decoupled from the Earth's rotation.

A well designed scene tree only couples together parts of the scene that move in the same way. Pivot objects are a good way to connect elements which have a common component to their movement but don't have a direct parent-child relationship.

Question 4:

Consider the world shown below, in world coordinates.



a. An orthographic camera with bounds (left = -1, right = 1, bottom = -1, top = 1) is placed at (2, 3).

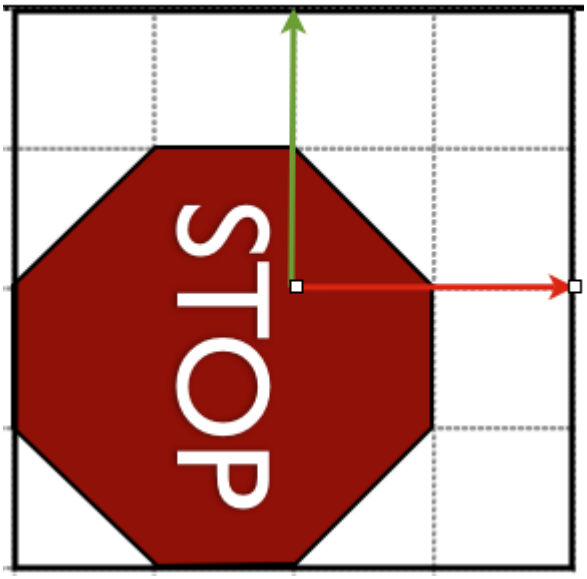
- What does it show?

- How would you compute the coordinate frame for the view transform?
  By default we get a camera with bounds (left = -1, right = 1, bottom = -1, top = 1) so we just have to construct a view transform with a translation. We must perform the *inverse* transformation since we are dealing with the camera.

```
//The inverse of translating by (2,3) is to translate by negative amounts.
CoordFrame2D frame = CoordFrame2D.identity()
    .translate(-2,-3); //translate in opposite directions
```

b. What if you placed it at (2,3), rotated the camera by 90 degrees then scale it by (2, 2).

- What does it show?



- How would you compute the coordinate frame for the view transform?

  We would need to perform the inverse of each transformation and also perform it in the reverse order, so to compute the view transform we would do

```
CoordFrame2D frame = CoordFrame2D.identity()
    .scale(0.5,0.5)    //scale by 1/scalefactor
    .rotate(-90);      //rotate in the opposite direction
    .translate(-2,-3); //translate in opposite directions
```