Note: please add the two jar package in project structure.

PartI:

The YMCA scene graph is made of 4 humanoid.
Every pose is a transformation using the humanoid.xml — rotate the left and right hand.

PartIIC:

How to control the keyboard:

left, right: node "No"
1, 2 : node "Yes"
up, down: move camera forward and backward
Y: switch to YMCA scene
R: switch to rooms scene

How to implement:

1. create a transform matrix for camera:

```
        private Matrix4f cameraTransform = new Matrix4f();
```

2. use the transform matrix to keep track of the accumulative rotations/translations instead of calculating the parameters of lookAt function():

```
        modelView.peek().mul(cameraTransform).lookAt(eye, look_at, up)
                .mul(trackballTransform);
```

3. alter the transform matrix according to keyboard events:

```
public void move_camera(float dir){ //forward & behind
    cameraTransform = new Matrix4f()
            .translate(0, 0, dir)
            .mul(cameraTransform);
}

public void node_no(float dir){
    cameraTransform = new Matrix4f()
            .rotate((float) Math.toRadians(dir * 10), 0, 1, 0)
            .mul(cameraTransform);
}

public void node_yes(float dir){
    cameraTransform = new Matrix4f()
            .rotate((float) Math.toRadians(dir * 10), 1, 0, 0)
```

```
        .mul(cameraTransform);
}
```

PartIII:

1. What is a scene graph made of? What are its various parts and what is the function of each part?
A scene graph is made of nodes(root node, group nodes, leaf nodes and transform nodes) and PolygonMesh.
And a node can have its name, material, transform, light.
Material and light affects the appearing of the object and transform affects the shape of the object. The heirarchy descends from the 'scene', which is the umbrella of everything in the xml file (importing models, as well as the placements of all the nodes under this umbrella).
The group node groups every individual or other groups of nodes under it together. They are containers that collect property, shape, and other group nodes into graphs. Transform nodes produce geometric transformations, including fields for scaling, rotating, and translating. Property nodes are used to indicate appearance and other physical characteristics of the location of nodes on the scene (such as color or texture). We imported all of the shapes that we used, and created them as instances of types of local variable shapes.

2. How is the scene graph drawn, and how does it ensure the correct transformation gets applied to the correct part?
From bottom to up, first draw every leaf node(draw the mesh) and apply transform to the node.
Then draw their parents and the transform. etc. It is a heirarchal system in which the nodes have a parent child relationship happening. Because it starts off in the broadest 'scene' category then goes into specify within group nodes, and within them, the shape nodes and property nodes. Basically every detail that entails transformations or properties or location of a node are nested within the heirarchy in such a way that you cannot confuse it as belonging to another object. Every nested node that is not encompassing as wide of a range of others can be drawn in a local space, then projected onto the bigger scene.

3. What is the use of the GL3ScenegraphRenderer class?
If you wanted to create a textual rendering of the scene graph (e.g. a textual description of each node) how would you write such a renderer?
GL3ScenegraphRenderer initial the shader program. It calls the root.draw() function to start drawing and actually draw the mesh(see drawMesh()).
Everyone has been set a name when read from xml file (see line 107 in SceneXMLReader.java — node.setName(name);)
So we can retrieve the node name when GL3ScenegraphRenderer.drawMesh() is

called
and implement drawText() similar to the class MyTextRenderer in JOGLFrame.java. The "sgraph" package basically can read an xml file using
sgraph.SceneXMLReader.importScenegraph() in SceneXMLReader.java
and parse it into a scene graph:
it reads in obj files and parses it into PolygonMesh and stores it
it reads in transforms and objects and store it as Nodes.