MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE

" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Victor Porev

Computer Graphics Programming

Laboratory Work 3

Coordinate transformations and projections. Animation. Control using input sensors

kulubecioglu mehmet

IM-14 FIOT

Class Number: 12

Kyiv

IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE

2025

1. Objective of the Project

The primary objective of this lab assignment is to build a three-dimensional scene using OpenGL ES 3.2 on Android and control it via user interactions. The goal is to gain hands-on experience in graphics programming and interactive rendering.

2. Technologies and Tools Used

- ❖ Java (Android development language)
- ❖ Android Studio
- ❖ OpenGL ES 3.2
- ❖ GLSurfaceView / Renderer structure
- ❖ Shader Programming (Vertex and Fragment Shader)
- ❖ Model-View-Projection (MVP) Matrix

3. Project Architecture

The project is designed with a modular and class-based architecture. Each class has a specific responsibility. Below is an overview of each class:

- **MainActivity.java**

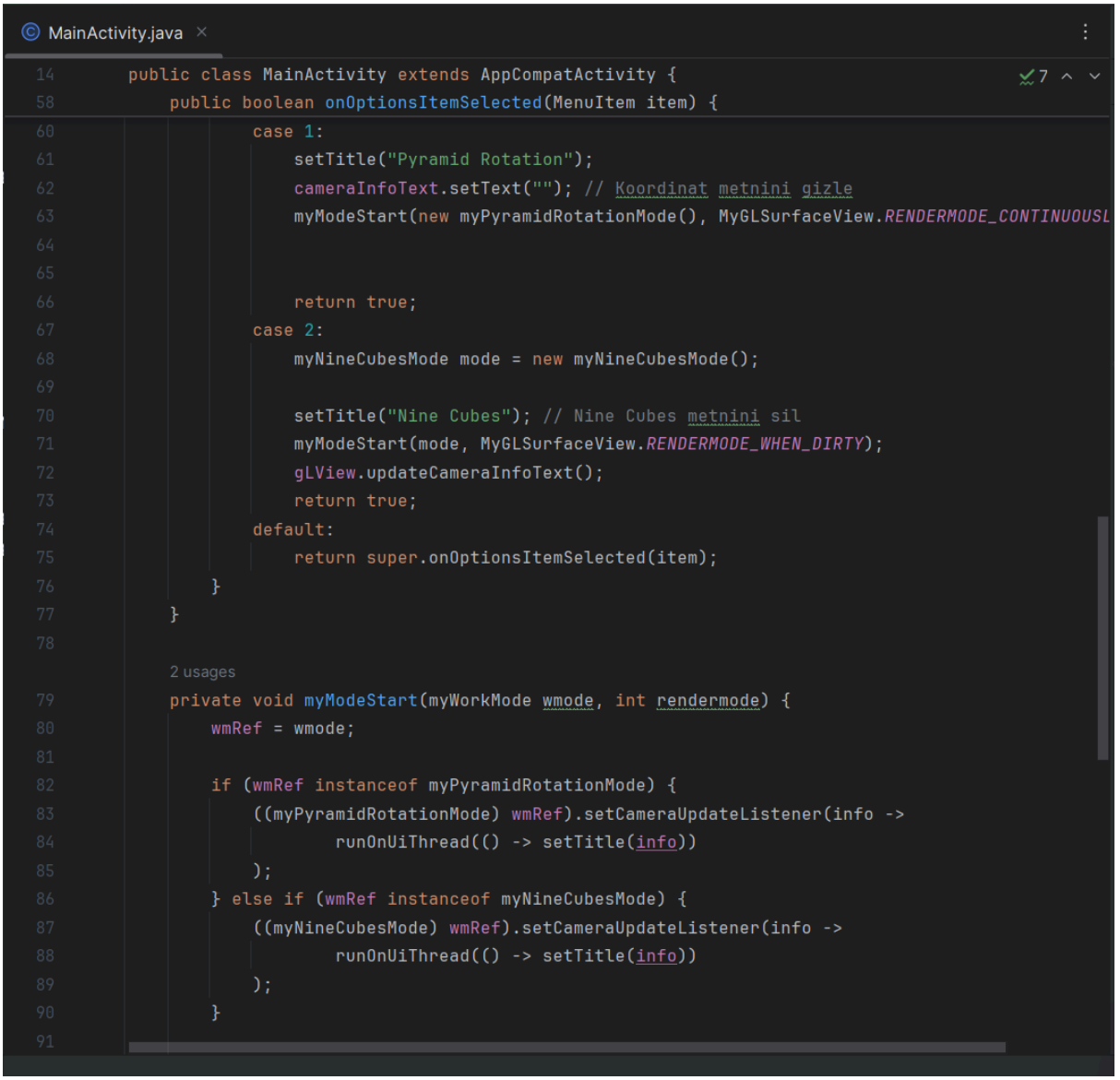
- ❖ Entry point of the application.
- ❖ Provides two scene options via menu: *Pyramid Rotation* and *Nine Cubes*.
- ❖ Updates camera coordinate information in the top title bar dynamically.
- ❖ Manages key events (arrow keys) for scene navigation in the cube scene.

MainActivity.java

```java
// Güncellenmis: MainActivity.java
package com.example.kulubecioglu_lab3_gles;

import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import androidx.appcompat.app.AppCompatActivity;
import android.view.KeyEvent;
import android.widget.FrameLayout;
import android.widget.TextView;


public class MainActivity extends AppCompatActivity {
    9 usages
    private MyGLSurfaceView gLView;

    7 usages
    private myWorkMode wmRef = null;
    8 usages
    private TextView cameraInfoText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);


        FrameLayout layout = new FrameLayout( context: this);
        gLView = new MyGLSurfaceView( context: this);

        cameraInfoText = new TextView( context: this);
        cameraInfoText.setText("a=0 b=0 x=0.0 y=0.0 z=0.0");
        cameraInfoText.setTextSize(14);
        cameraInfoText.setTextColor(0xFFFFFFFF);
```

MainActivity.java

```
14      public class MainActivity extends AppCompatActivity {                    7
21          protected void onCreate(Bundle savedInstanceState) {
30              cameraInfoText.setTextSize(14);
31              cameraInfoText.setTextColor(0xFFFFFFFF);
32
33
34              FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(
35                      FrameLayout.LayoutParams.WRAP_CONTENT,
36                      FrameLayout.LayoutParams.WRAP_CONTENT
37              );
38              params.gravity = Gravity.TOP | Gravity.CENTER_HORIZONTAL;
39              params.topMargin = 10;
40              cameraInfoText.setLayoutParams(params);
41
42              layout.addView(gLView);
43              layout.addView(cameraInfoText);
44
45              setContentView(layout);
46
47              gLView.setCameraInfoTextView(cameraInfoText);
48          }
49
50          @Override
51          public boolean onCreateOptionsMenu(Menu menu) {
52              menu.add( groupId: 0,  itemId: 1,  order: 0,  title: "Pyramid Rotation");
53              menu.add( groupId: 0,  itemId: 2,  order: 0,  title: "Nine Cubes");
54              return true;
55          }
56
57          @Override
58          public boolean onOptionsItemSelected(MenuItem item) {
59              switch (item.getItemId()) {
60                  case 1:
61                      setTitle("Pyramid Rotation");
62                      cameraInfoText.setText(""); // Koordinat metnini gizle
```


MainActivity.java
public class MainActivity extends AppCompatActivity {
public boolean onOptionsItemSelected(MenuItem item) {
case 1:
setTitle("Pyramid Rotation");
cameraInfoText.setText(""); // Koordinat metnini gizle
myModeStart(new myPyramidRotationMode(), MyGLSurfaceView.RENDERMODE_CONTINUOUSL
return true;
case 2:
myNineCubesMode mode = new myNineCubesMode();
setTitle("Nine Cubes"); // Nine Cubes metnini sil
myModeStart(mode, MyGLSurfaceView.RENDERMODE_WHEN_DIRTY);
gLView.updateCameraInfoText();
return true;
default:
return super.onOptionsItemSelected(item);
}
}
2 usages
private void myModeStart(myWorkMode wmode, int rendermode) {
wmRef = wmode;
if (wmRef instanceof myPyramidRotationMode) {
((myPyramidRotationMode) wmRef).setCameraUpdateListener(info ->
runOnUiThread(() -> setTitle(info))
);
} else if (wmRef instanceof myNineCubesMode) {
((myNineCubesMode) wmRef).setCameraUpdateListener(info ->
runOnUiThread(() -> setTitle(info))
);
}

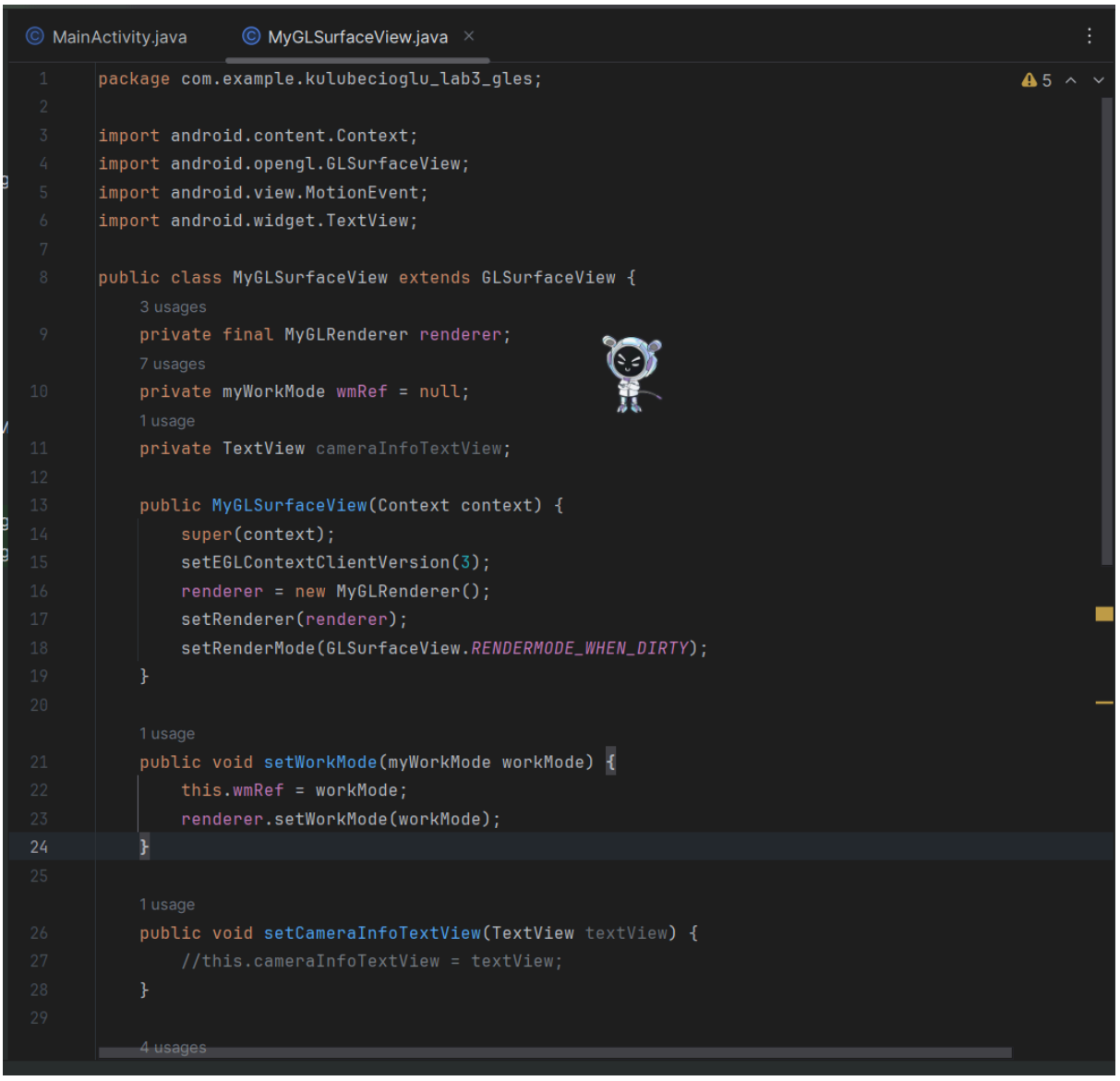
MainActivity.java

```
14      public class MainActivity extends AppCompatActivity {                    7
79          private void myModeStart(myWorkMode wmode, int rendermode) {
89                  );
90              }
91
92              gLView.setWorkMode(wmode);
93              gLView.setRenderMode(rendermode);
94              gLView.requestRender();
95          }
96
97
98
99          @Override
100         public boolean onKeyDown(int keyCode, KeyEvent event) {
101
102             if (wmRef instanceof myNineCubesMode) {
103                 myNineCubesMode mode = (myNineCubesMode) wmRef;
104
105
106                 switch (keyCode) {
107                     case KeyEvent.KEYCODE_DPAD_LEFT:
108                         mode.adjustAlpha( delta: -5);
109                         break;
110                     case KeyEvent.KEYCODE_DPAD_RIGHT:
111                         mode.adjustAlpha( delta: 5);
112                         break;
113                     case KeyEvent.KEYCODE_DPAD_UP:
114                         mode.adjustBeta( delta: 5);
115                         break;
116                     case KeyEvent.KEYCODE_DPAD_DOWN:
117                         mode.adjustBeta( delta: -5);
118                         break;
119                 }
120                 gLView.updateCameraInfoText();
121                 gLView.requestRender();
```

```
121                 gLView.requestRender();
122
123                 return true;
124             }
125             return super.onKeyDown(keyCode, event);
126         }
127     }
```

- **MyGLSurfaceView.java**

❖ Custom class extending **`GLSurfaceView`.**
❖ Captures touch gestures and forwards them to the active scene.
❖ Updates camera coordinate information in a **`TextView`.**



MainActivity.java MyGLSurfaceView.java

```
8   public class MyGLSurfaceView extends GLSurfaceView {                    5
22      this.wmRef = workMode;
23      renderer.setWorkMode(workMode);
24  }
25
        1 usage
26  public void setCameraInfoTextView(TextView textView) {
27      //this.cameraInfoTextView = textView;
28  }
29
        4 usages
30  public void updateCameraInfoText() {
31      if (wmRef instanceof myNineCubesMode && cameraInfoTextView != null) {
32          myNineCubesMode mode = (myNineCubesMode) wmRef;
33          String info = String.format("a=%.0f b=%.0f x=%.1f y=%.1f z=%.1f",
34                  mode.getAlpha(), mode.getBeta(), mode.getCamX(), mode.getCamY(), mode.getCamZ(
35          //cameraInfoTextView.setText(info);
36      }
37  }
38
39  @Override
40  public boolean onTouchEvent(MotionEvent e) {
41      if (wmRef == null || wmRef.onTouchNotUsed()) return false;
42
43      int cx = getWidth();
44      int cy = getHeight();
45      float x = e.getX();
46      float y = e.getY();
47
48      switch (e.getAction()) {
49          case MotionEvent.ACTION_DOWN:
50              if (wmRef.onActionDown(x, y, cx, cy)) updateCameraInfoText();
51              break;
52          case MotionEvent.ACTION_MOVE:
53              if (wmRef.onActionMove(x, y, cx, cy)) updateCameraInfoText();
```

```
53              if (wmRef.onActionMove(x, y, cx, cy)) updateCameraInfoText();
54              break;
55      }
56      requestRender();
57      return true;
58  }
59  }
60
```

- **MyGLRenderer.java**

❖ Handles the rendering operations.

❖ Controls rendering flow via **`onSurfaceCreated`**, **`onDrawFrame`**, and **`onSurfaceChanged`.**

❖ Draws scenes based on the active **`myWorkMode`.**

MainActivity.java | MyGLSurfaceView.java | MyGLRenderer.java

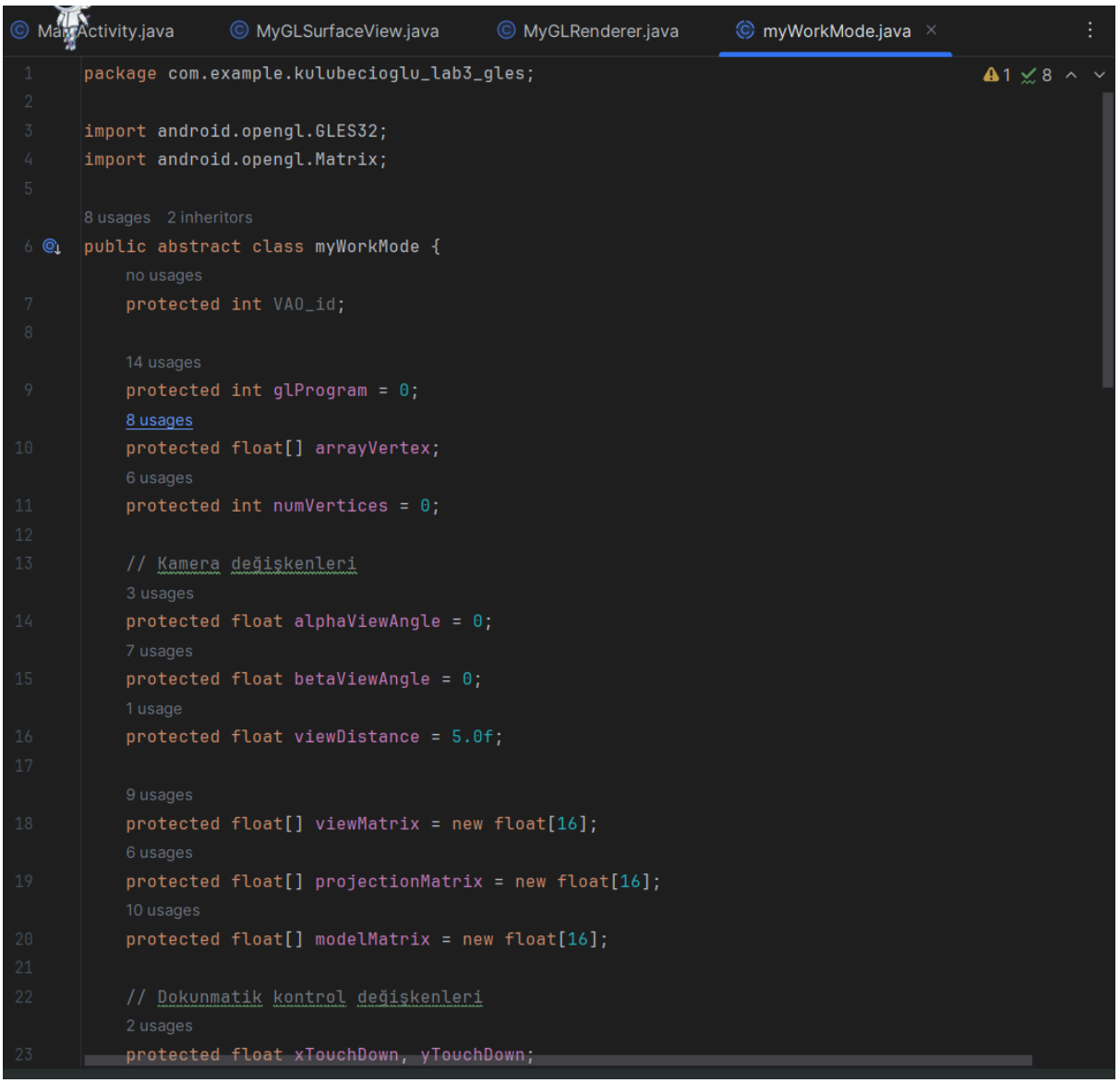
```
1   package com.example.kulubecioglu_lab3_gles;
2
3   import android.opengl.GLES32;
4   import android.opengl.GLSurfaceView;
5
6   import javax.microedition.khronos.egl.EGLConfig;
7   import javax.microedition.khronos.opengles.GL10;
8
    2 usages
9   public class MyGLRenderer implements GLSurfaceView.Renderer {
        7 usages
10      private myWorkMode wmRef = null;
        2 usages
11      private int renderWidth = 1;
        2 usages
12      private int renderHeight = 1;
13
        1 usage
14      public void setWorkMode(myWorkMode workMode) {
15          this.wmRef = workMode;
16      }
17
        1 usage
18      @Override
19      public void onSurfaceCreated(GL10 gl, EGLConfig config) {
20          GLES32.glEnable(GLES32.GL_DEPTH_TEST);
21          GLES32.glClearColor( red: 0.1f,  green: 0.1f,  blue: 0.1f,  alpha: 1.0f);
22      }
23
        no usages
24      @Override
25      public void onSurfaceChanged(GL10 gl, int width, int height) {
26          GLES32.glViewport( x: 0,  y: 0, width, height);
27          renderWidth = width;
28          renderHeight = height;
```

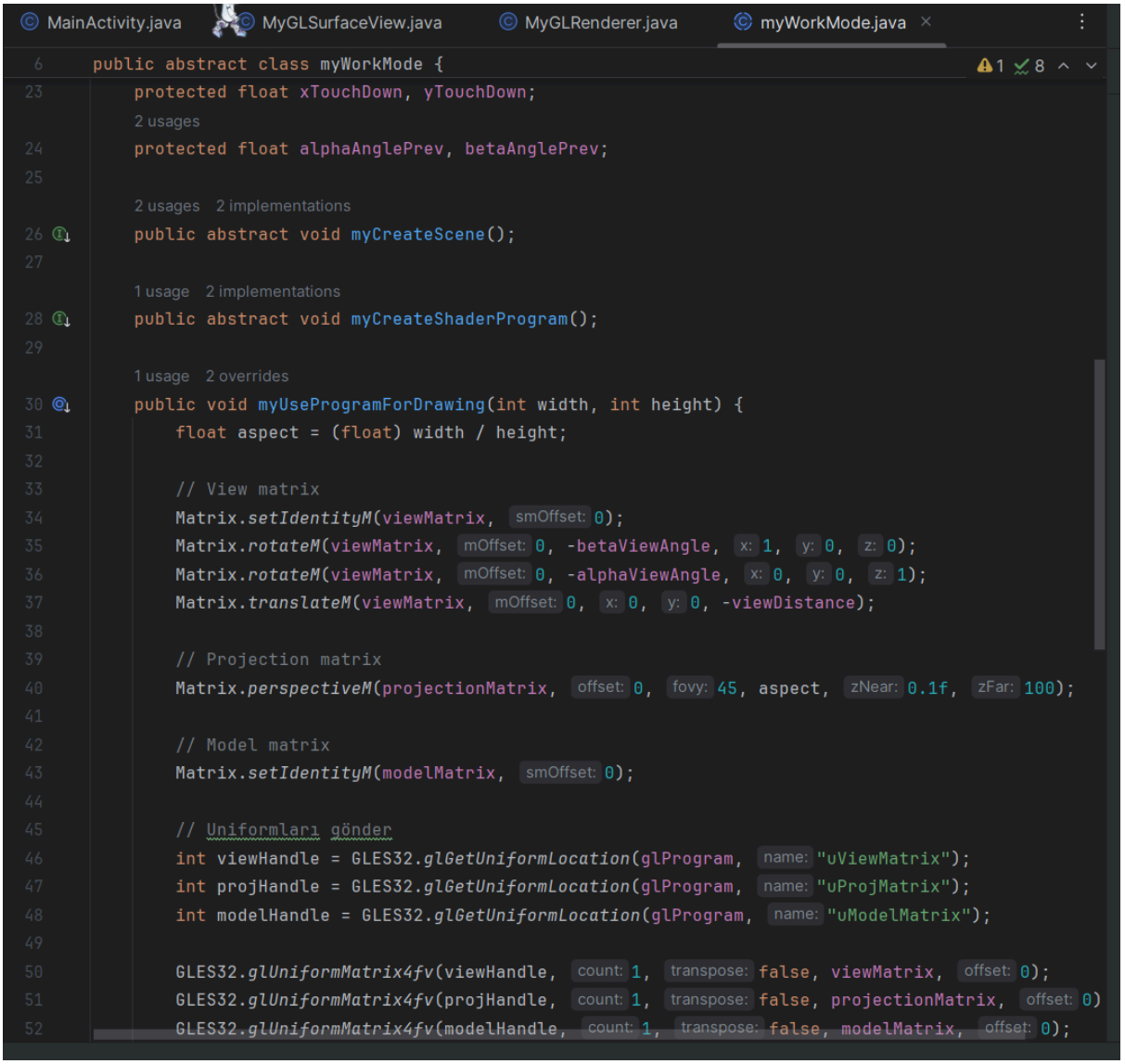
```
28          renderHeight = height;
29      }
30
        no usages
31      @Override
32      public void onDrawFrame(GL10 gl) {
33          GLES32.glClear( mask: GLES32.GL_COLOR_BUFFER_BIT | GLES32.GL_DEPTH_BUFFER_BIT);
34
35          if (wmRef == null) return;
36
37          if (wmRef.getProgramId() == 0) wmRef.myCreateShaderProgram();
38          if (wmRef.getProgramId() == 0) return;
39
40          GLES32.glUseProgram(wmRef.getProgramId());
41          wmRef.myUseProgramForDrawing(renderWidth, renderHeight);
42      }
43  }
44
```

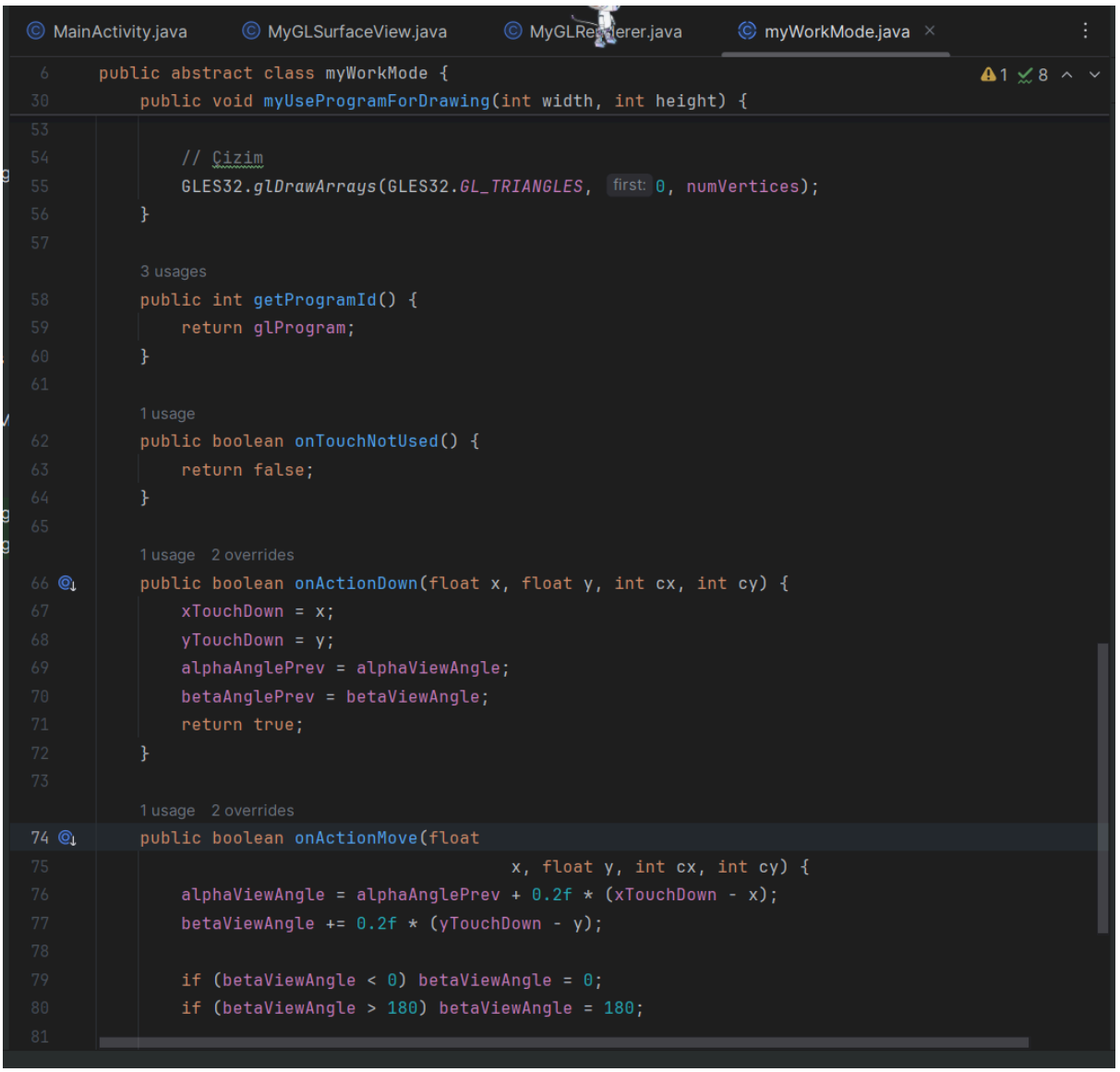
- **myWorkMode.java**

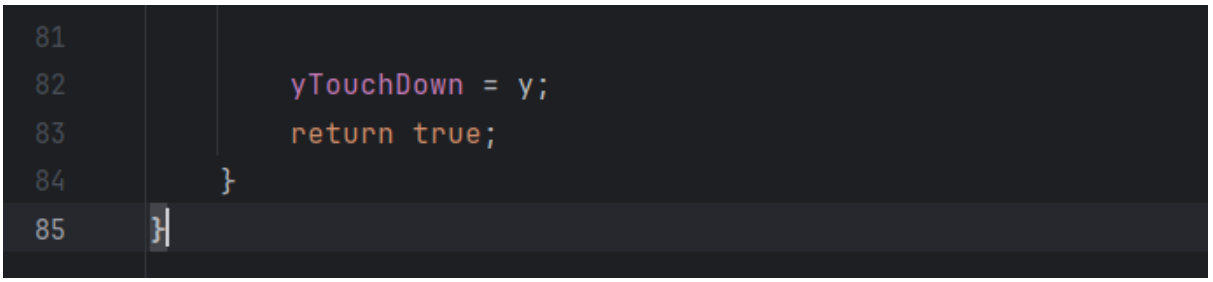
❖ Abstract base class for all scene modes.

❖ Provides template methods like **`myCreateScene()`**, **`myUseProgramForDrawing()`**.


MainActivity.java
MyGLSurfaceView.java
MyGLRenderer.java
myWorkMode.java
1 8
package com.example.kulubecioglu_lab3_gles;
import android.opengl.GLES32;
import android.opengl.Matrix;
8 usages 2 inheritors
public abstract class myWorkMode {
no usages
protected int VAO_id;
14 usages
protected int glProgram = 0;
8 usages
protected float[] arrayVertex;
6 usages
protected int numVertices = 0;
// Kamera değişkenleri
3 usages
protected float alphaViewAngle = 0;
7 usages
protected float betaViewAngle = 0;
1 usage
protected float viewDistance = 5.0f;
9 usages
protected float[] viewMatrix = new float[16];
6 usages
protected float[] projectionMatrix = new float[16];
10 usages
protected float[] modelMatrix = new float[16];
// Dokunmatik kontrol değişkenleri
2 usages
protected float xTouchDown, yTouchDown;

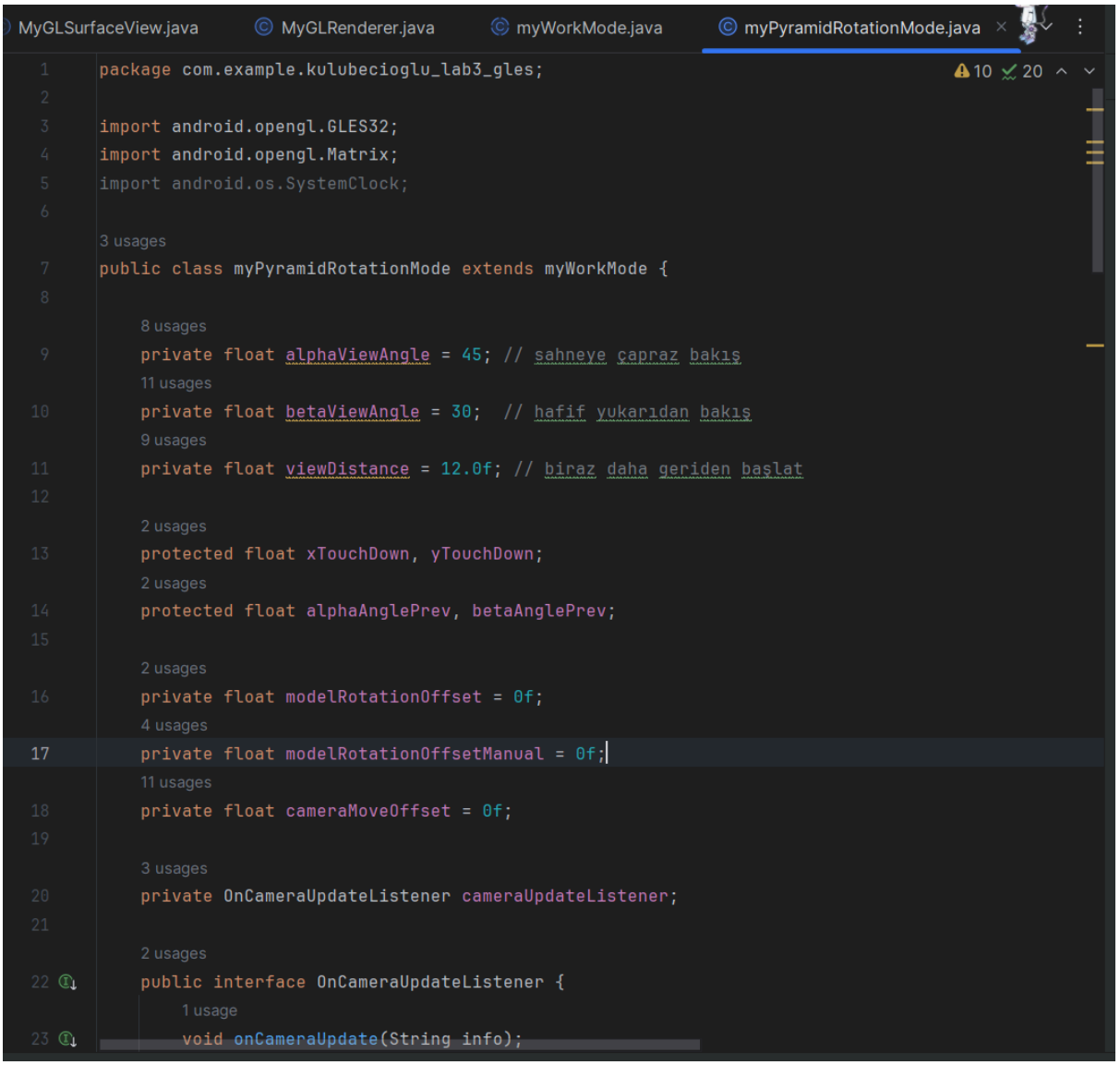

MainActivity.java
MyGLSurfaceView.java
MyGLRenderer.java
myWorkMode.java
public abstract class myWorkMode {
protected float xTouchDown, yTouchDown;
2 usages
protected float alphaAnglePrev, betaAnglePrev;
2 usages 2 implementations
public abstract void myCreateScene();
1 usage 2 implementations
public abstract void myCreateShaderProgram();
1 usage 2 overrides
public void myUseProgramForDrawing(int width, int height) {
float aspect = (float) width / height;
// View matrix
Matrix.setIdentityM(viewMatrix, smOffset: 0);
Matrix.rotateM(viewMatrix, mOffset: 0, -betaViewAngle, x: 1, y: 0, z: 0);
Matrix.rotateM(viewMatrix, mOffset: 0, -alphaViewAngle, x: 0, y: 0, z: 1);
Matrix.translateM(viewMatrix, mOffset: 0, x: 0, y: 0, -viewDistance);
// Projection matrix
Matrix.perspectiveM(projectionMatrix, offset: 0, fovy: 45, aspect, zNear: 0.1f, zFar: 100);
// Model matrix
Matrix.setIdentityM(modelMatrix, smOffset: 0);
// Uniformları gönder
int viewHandle = GLES32.glGetUniformLocation(glProgram, name: "uViewMatrix");
int projHandle = GLES32.glGetUniformLocation(glProgram, name: "uProjMatrix");
int modelHandle = GLES32.glGetUniformLocation(glProgram, name: "uModelMatrix");
GLES32.glUniformMatrix4fv(viewHandle, count: 1, transpose: false, viewMatrix, offset: 0);
GLES32.glUniformMatrix4fv(projHandle, count: 1, transpose: false, projectionMatrix, offset: 0)
GLES32.glUniformMatrix4fv(modelHandle, count: 1, transpose: false, modelMatrix, offset: 0);

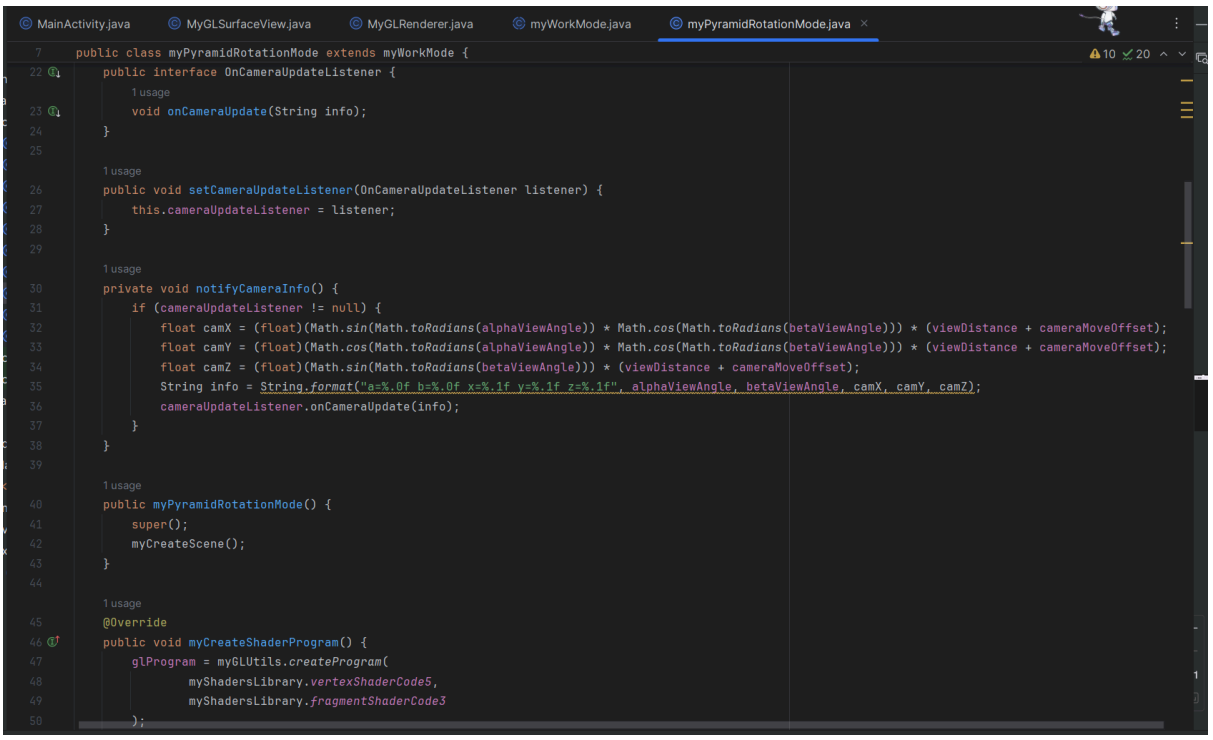

MainActivity.java
MyGLSurfaceView.java
MyGLRenderer.java
myWorkMode.java
public abstract class myWorkMode {
public void myUseProgramForDrawing(int width, int height) {
// Cizim
GLES32.glDrawArrays(GLES32.GL_TRIANGLES, first: 0, numVertices);
}
3 usages
public int getProgramId() {
return glProgram;
}
1 usage
public boolean onTouchNotUsed() {
return false;
}
1 usage 2 overrides
public boolean onActionDown(float x, float y, int cx, int cy) {
xTouchDown = x;
yTouchDown = y;
alphaAnglePrev = alphaViewAngle;
betaAnglePrev = betaViewAngle;
return true;
}
1 usage 2 overrides
public boolean onActionMove(float
x, float y, int cx, int cy) {
alphaViewAngle = alphaAnglePrev + 0.2f * (xTouchDown - x);
betaViewAngle += 0.2f * (yTouchDown - y);
if (betaViewAngle < 0) betaViewAngle = 0;
if (betaViewAngle > 180) betaViewAngle = 180;


yTouchDown = y;
return true;
}
}

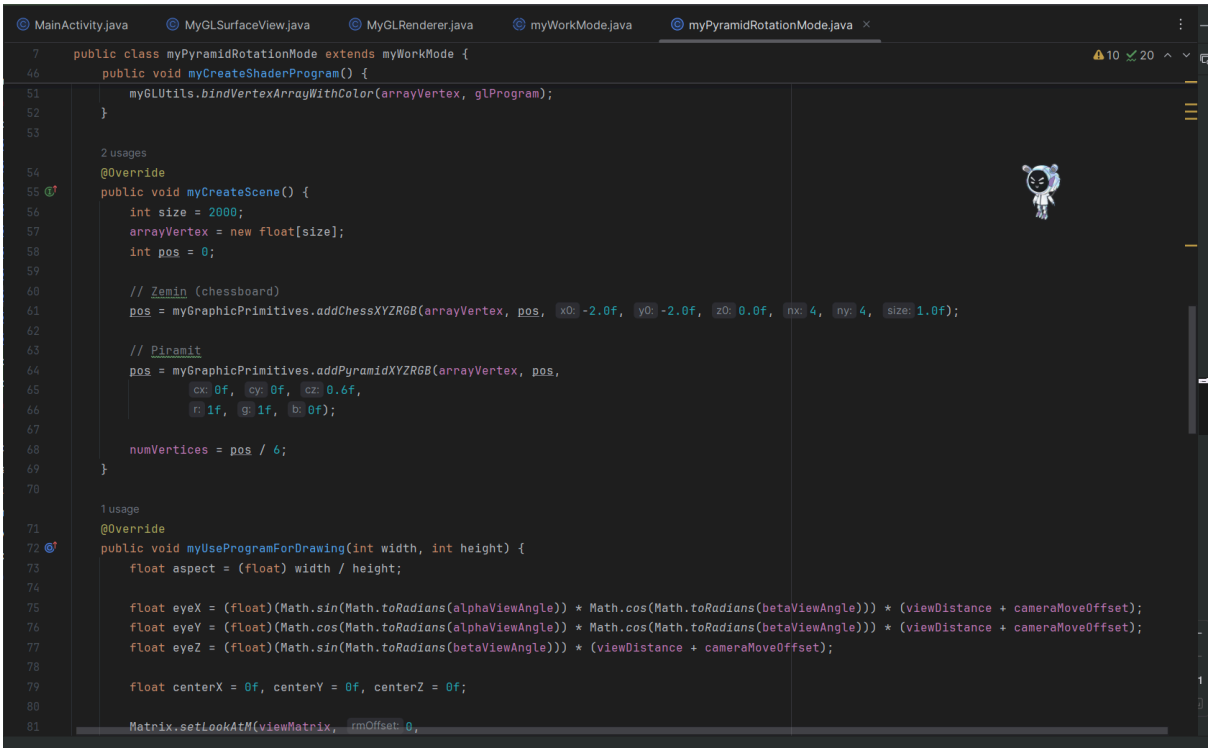
- **myPyramidRotationMode.java**

❖ Implements the pyramid scene.
❖ Camera is positioned diagonally.
❖ The pyramid continuously rotates on its own axis.
❖ User can rotate the scene or zoom in/out using touch gestures.
❖ Camera information is updated dynamically and displayed in the UI.





```
MainActivity.java  MyGLSurfaceView.java  MyGLRenderer.java  myWorkMode.java  myPyramidRotationMode.java

public class myPyramidRotationMode extends myWorkMode {
    public interface OnCameraUpdateListener {
        1 usage
        void onCameraUpdate(String info);
    }

    1 usage
    public void setCameraUpdateListener(OnCameraUpdateListener listener) {
        this.cameraUpdateListener = listener;
    }

    1 usage
    private void notifyCameraInfo() {
        if (cameraUpdateListener != null) {
            float camX = (float)(Math.sin(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
            float camY = (float)(Math.cos(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
            float camZ = (float)(Math.sin(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
            String info = String.format("a=%.0f b=%.0f x=%.1f y=%.1f z=%.1f", alphaViewAngle, betaViewAngle, camX, camY, camZ);
            cameraUpdateListener.onCameraUpdate(info);
        }
    }

    1 usage
    public myPyramidRotationMode() {
        super();
        myCreateScene();
    }

    1 usage
    @Override
    public void myCreateShaderProgram() {
        glProgram = myGLUtils.createProgram(
                myShadersLibrary.vertexShaderCode5,
                myShadersLibrary.fragmentShaderCode3
        );
```



```
MainActivity.java  MyGLSurfaceView.java  MyGLRenderer.java  myWorkMode.java  myPyramidRotationMode.java

public class myPyramidRotationMode extends myWorkMode {
    public void myCreateShaderProgram() {
        myGLUtils.bindVertexArrayWithColor(arrayVertex, glProgram);
    }

    2 usages
    @Override
    public void myCreateScene() {
        int size = 2000;
        arrayVertex = new float[size];
        int pos = 0;

        // Zemin (chessboard)
        pos = myGraphicPrimitives.addChessXYZRGB(arrayVertex, pos, x0: -2.0f, y0: -2.0f, z0: 0.0f, nx: 4, ny: 4, size: 1.0f);

        // Piramit
        pos = myGraphicPrimitives.addPyramidXYZRGB(arrayVertex, pos,
                cx: 0f, cy: 0f, cz: 0.6f,
                r: 1f, g: 1f, b: 0f);

        numVertices = pos / 6;
    }

    1 usage
    @Override
    public void myUseProgramForDrawing(int width, int height) {
        float aspect = (float) width / height;

        float eyeX = (float)(Math.sin(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
        float eyeY = (float)(Math.cos(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
        float eyeZ = (float)(Math.sin(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);

        float centerX = 0f, centerY = 0f, centerZ = 0f;

        Matrix.setLookAtM(viewMatrix, rmOffset: 0,
```

MainActivity.java | MyGLSurfaceView.java | MyGLRenderer.java | myWorkMode.java | myPyramidRotationMode.java

```
public class myPyramidRotationMode extends myWorkMode {
    public void myUseProgramForDrawing(int width, int height) {
                eyeX, eyeY, eyeZ,
                centerX, centerY, centerZ,
                upX: 0f, upY: 0f, upZ: 1f);

        Matrix.perspectiveM(projectionMatrix, offset: 0, fovy: 45, aspect, zNear: 0.1f, zFar: 30);

        int viewHandle = GLES32.glGetUniformLocation(glProgram, name: "uViewMatrix");
        int projHandle = GLES32.glGetUniformLocation(glProgram, name: "uProjMatrix");
        int modelHandle = GLES32.glGetUniformLocation(glProgram, name: "uModelMatrix");

        // Zemin çizimi (modelRotationOffsetManual dönüşüyle)
        Matrix.setIdentityM(modelMatrix, smOffset: 0);
        Matrix.rotateM(modelMatrix, mOffset: 0, modelRotationOffsetManual, x: 0, y: 0, z: 1);
        GLES32.glUniformMatrix4fv(modelHandle, count: 1, transpose: false, modelMatrix, offset: 0);
        GLES32.glUniformMatrix4fv(viewHandle, count: 1, transpose: false, viewMatrix, offset: 0);
        GLES32.glUniformMatrix4fv(projHandle, count: 1, transpose: false, projectionMatrix, offset: 0);
        GLES32.glDrawArrays(GLES32.GL_TRIANGLES, first: 0, count: numVertices - 12);

        // Piramit çizimi (otomatik dönüş + manuel dönüş)
        Matrix.setIdentityM(modelMatrix, smOffset: 0);
        modelRotationOffset += 7.2f;
        Matrix.rotateM(modelMatrix, mOffset: 0, a: modelRotationOffset + modelRotationOffsetManual, x: 0, y: 0, z: 1);
        GLES32.glUniformMatrix4fv(modelHandle, count: 1, transpose: false, modelMatrix, offset: 0);
        GLES32.glDrawArrays(GLES32.GL_TRIANGLES, first: numVertices - 12, count: 12);

        notifyCameraInfo();
    }

    1 usage
    @Override
    public boolean onActionDown(float x, float y, int cx, int cy) {
        xTouchDown = x;
        yTouchDown = y;
```

MainActivity.java | MyGLSurfaceView.java | MyGLRenderer.java | myWorkMode.java | myPyramidRotationMode.java

```
public class myPyramidRotationMode extends myWorkMode {
    public boolean onActionDown(float x, float y, int cx, int cy) {
        alphaAnglePrev = modelRotationOffsetManual;
        betaAnglePrev = cameraMoveOffset;
        return true;
    }

    1 usage
    @Override
    public boolean onActionMove(float x, float y, int cx, int cy) {
        float dx = x - xTouchDown;
        float dy = y - yTouchDown;

        modelRotationOffsetManual = alphaAnglePrev + dx * 0.02f;
        cameraMoveOffset = betaAnglePrev - dy * 0.005f;

        return true;
    }

    no usages
    public float getAlpha() {
        return alphaViewAngle;
    }

    no usages
    public float getBeta() {
        return betaViewAngle;
    }

    no usages
    public float getCamX() {
        return (float)(Math.sin(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
    }

    no usages
```

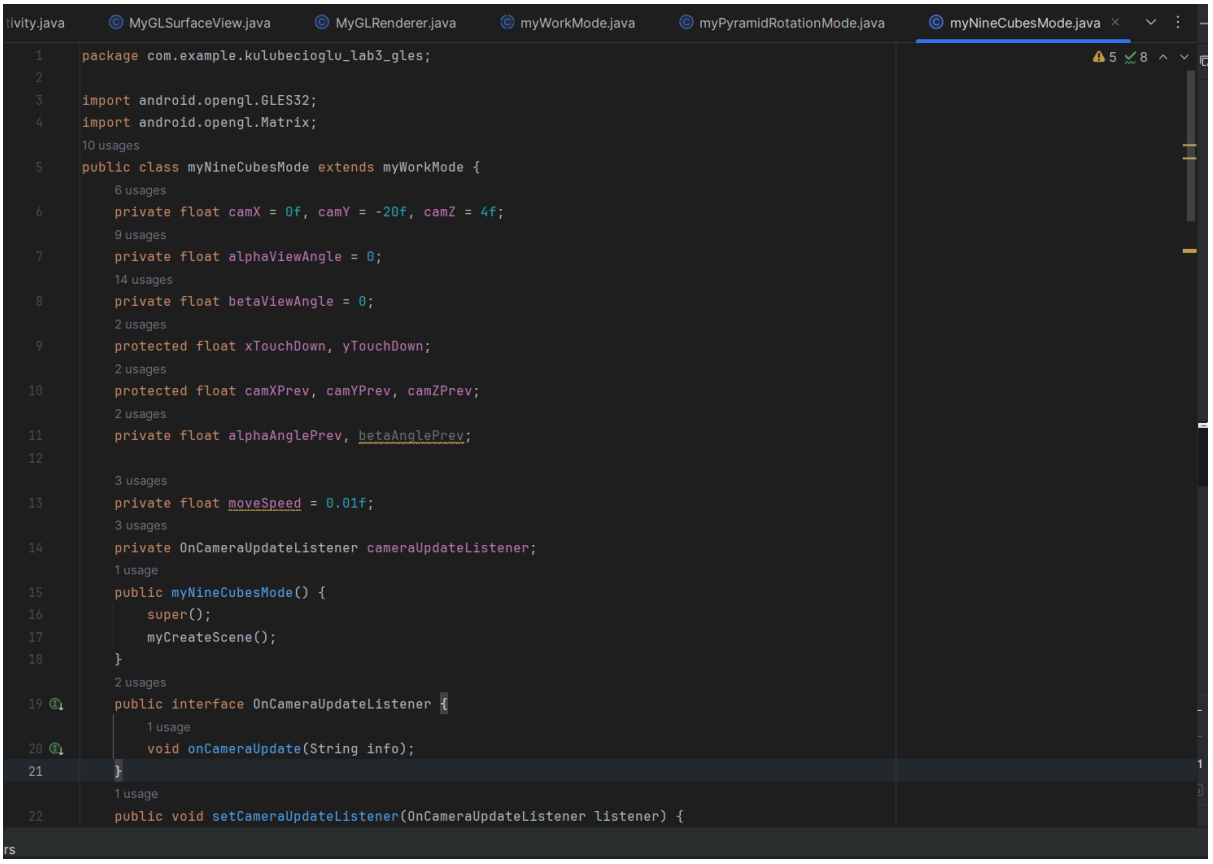
```
141
        no usages
142     public float getCamY() {
143         return (float)(Math.cos(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
144     }
145
        no usages
146     public float getCamZ() {
147         return (float)(Math.sin(Math.toRadians(betaViewAngle))) * (viewDistance + cameraMoveOffset);
148     }
149 }
```

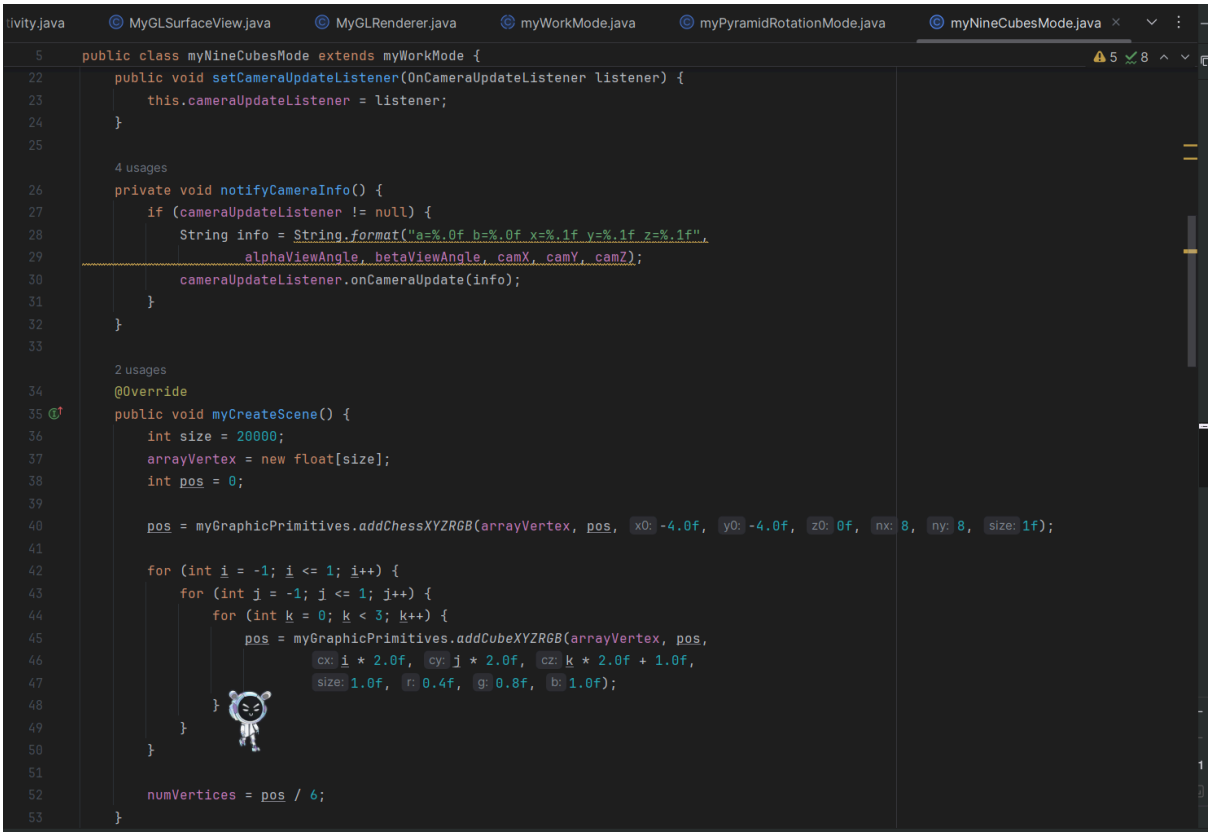
- **myNineCubesMode.java**

❖ Implements a 3x3x3 cube grid scene.
❖ Provides free camera movement inside the scene.
❖ Navigation via touch gestures and arrow keys.
❖ Includes a chessboard-style floor for spatial reference.

- myGraphicPrimitives.java

❖ Helper class for adding geometric shapes (cubes, pyramids, floor).
❖ Includes methods like **`addCubeXYZRGB()`**, **`addPyramidXYZRGB()`**, **`addChessXYZRGB()`**.

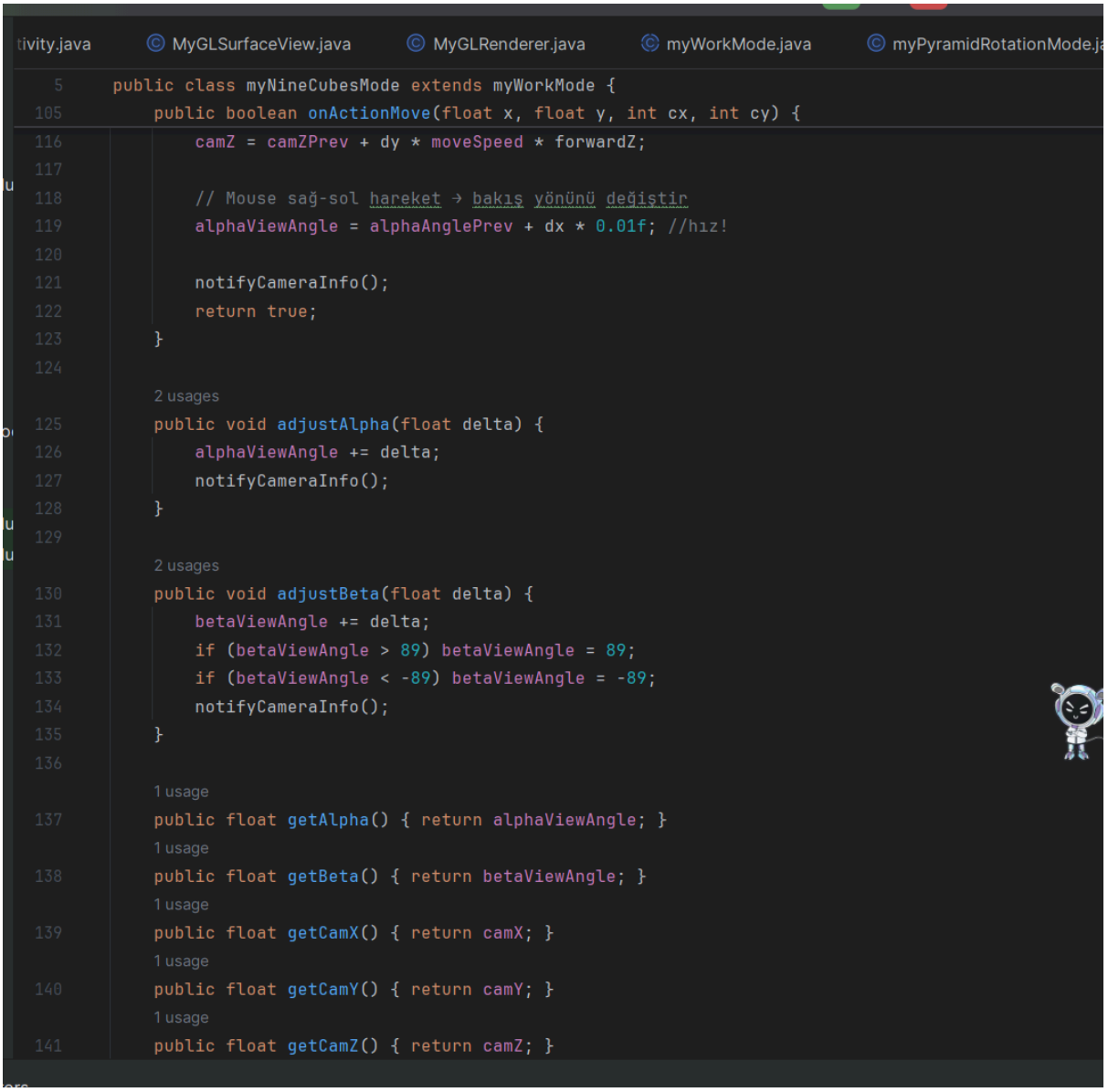

tivity.java
MyGLSurfaceView.java
MyGLRenderer.java
myWorkMode.java
myPyramidRotationMode.java
myNineCubesMode.java
5 8
package com.example.kulubecioglu_lab3_gles;
import android.opengl.GLES32;
import android.opengl.Matrix;
10 usages
public class myNineCubesMode extends myWorkMode {
6 usages
private float camX = 0f, camY = -20f, camZ = 4f;
9 usages
private float alphaViewAngle = 0;
14 usages
private float betaViewAngle = 0;
2 usages
protected float xTouchDown, yTouchDown;
2 usages
protected float camXPrev, camYPrev, camZPrev;
2 usages
private float alphaAnglePrev, betaAnglePrev;
3 usages
private float moveSpeed = 0.01f;
3 usages
private OnCameraUpdateListener cameraUpdateListener;
1 usage
public myNineCubesMode() {
super();
myCreateScene();
}
2 usages
public interface OnCameraUpdateListener {
1 usage
void onCameraUpdate(String info);
}
1 usage
public void setCameraUpdateListener(OnCameraUpdateListener listener) {
rs


tivity.java
MyGLSurfaceView.java
MyGLRenderer.java
myWorkMode.java
myPyramidRotationMode.java
myNineCubesMode.java
5 8
public class myNineCubesMode extends myWorkMode {
public void setCameraUpdateListener(OnCameraUpdateListener listener) {
this.cameraUpdateListener = listener;
}
4 usages
private void notifyCameraInfo() {
if (cameraUpdateListener != null) {
String info = String.format("a=%.0f b=%.0f x=%.1f y=%.1f z=%.1f",
alphaViewAngle, betaViewAngle, camX, camY, camZ);
cameraUpdateListener.onCameraUpdate(info);
}
}
2 usages
@Override
public void myCreateScene() {
int size = 20000;
arrayVertex = new float[size];
int pos = 0;
pos = myGraphicPrimitives.addChessXYZRGB(arrayVertex, pos, x0: -4.0f, y0: -4.0f, z0: 0f, nx: 8, ny: 8, size: 1f);
for (int i = -1; i <= 1; i++) {
for (int j = -1; j <= 1; j++) {
for (int k = 0; k < 3; k++) {
pos = myGraphicPrimitives.addCubeXYZRGB(arrayVertex, pos,
cx: i * 2.0f, cy: j * 2.0f, cz: k * 2.0f + 1.0f,
size: 1.0f, r: 0.4f, g: 0.8f, b: 1.0f);
}
}
}
numVertices = pos / 6;
}

```
tivity.java  MyGLSurfaceView.java  MyGLRenderer.java  myWorkMode.java  myPyramidRotationMode.java  myNineCubesMode.java
```

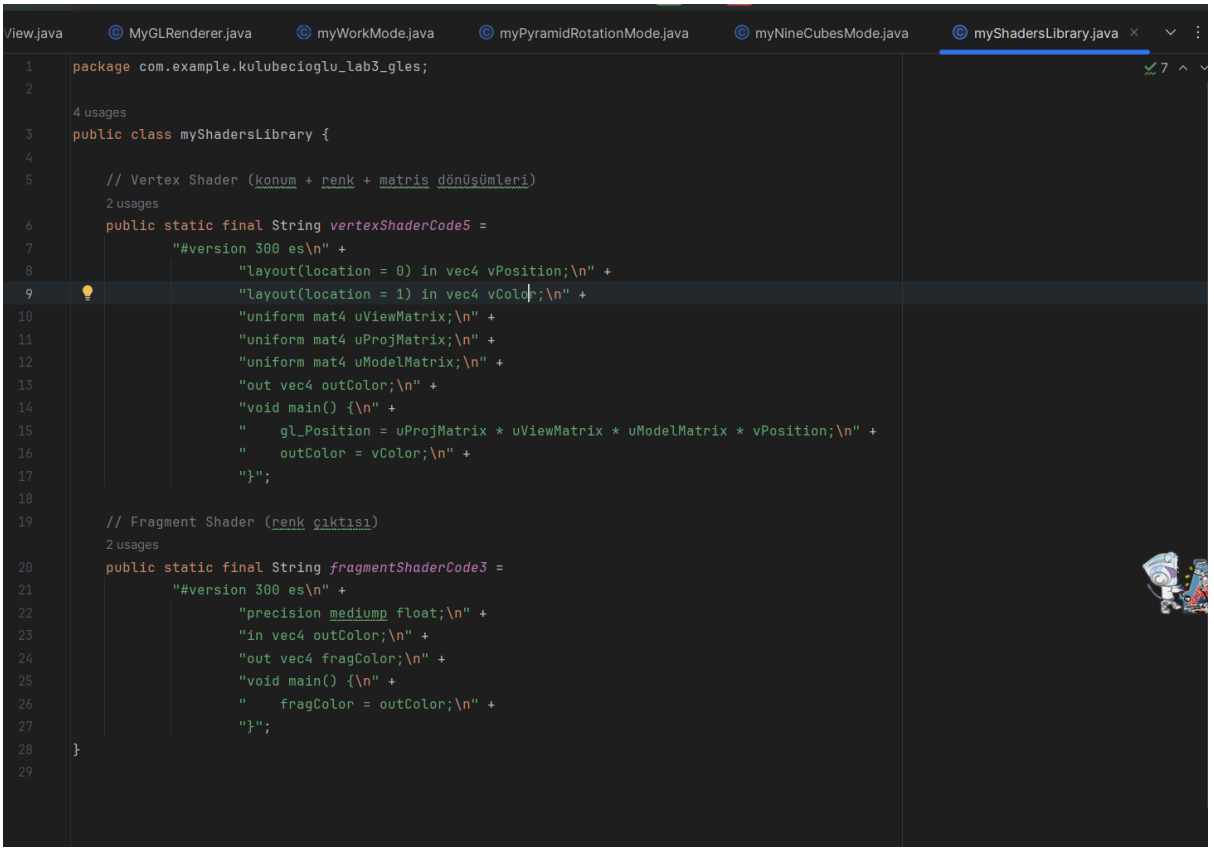
```
5   public class myNineCubesMode extends myWorkMode {                                  ⚠ 5  ✓ 8
53      }
54
        1 usage
55      @Override
56      public void myCreateShaderProgram() {
57          glProgram = myGLUtils.createProgram(
58                  myShadersLibrary.vertexShaderCode5,
59                  myShadersLibrary.fragmentShaderCode3
60          );
61          myGLUtils.bindVertexArrayWithColor(arrayVertex, glProgram);
62      }
63
        1 usage
64      @Override
65      public void myUseProgramForDrawing(int width, int height) {
66          float aspect = (float) width / height;
67
68          float lookX = camX + (float)(Math.sin(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle)));
69          float lookY = camY + (float)(Math.cos(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle)));
70          float lookZ = camZ + (float)(Math.sin(Math.toRadians(betaViewAngle)));
71
72          Matrix.setLookAtM(viewMatrix, rmOffset: 0,
73                  camX, camY, camZ,
74                  lookX, lookY, lookZ,
75                  upX: 0f, upY: 0f, upZ: 1f);
76
77          Matrix.perspectiveM(projectionMatrix, offset: 0, fovy: 45, aspect, zNear: 0.1f, zFar: 100);
78
79          int viewHandle = GLES32.glGetUniformLocation(glProgram, name: "uViewMatrix");
80          int projHandle = GLES32.glGetUniformLocation(glProgram, name: "uProjMatrix");
81          int modelHandle = GLES32.glGetUniformLocation(glProgram, name: "uModelMatrix");
82
83          Matrix.setIdentityM(modelMatrix, smOffset: 0);
84          GLES32.glUniformMatrix4fv(viewHandle, count: 1, transpose: false, viewMatrix, offset: 0);
```

```
85          GLES32.glUniformMatrix4fv(projHandle, count: 1, transpose: false, projectionMatrix, offset: 0);
86          GLES32.glUniformMatrix4fv(modelHandle, count: 1, transpose: false, modelMatrix, offset: 0);
87
88          GLES32.glDrawArrays(GLES32.GL_TRIANGLES, first: 0, numVertices);
89          notifyCameraInfo();
90      }
91
        1 usage
92      @Override
93      public boolean onActionDown(float x, float y, int cx, int cy) {
94          xTouchDown = x;
95          yTouchDown = y;
96          camXPrev = camX;
97          camYPrev = camY;
98          camZPrev = camZ;
99          alphaAnglePrev = alphaViewAngle;
100         betaAnglePrev = betaViewAngle;
101         return true;
102     }
103
        1 usage
104     @Override
105     public boolean onActionMove(float x, float y, int cx, int cy) {
106         float dx = x - xTouchDown;
107         float dy = y - yTouchDown;
108
109         // Mouse ileri-geri hareket → kamera ileri-geri
110         float forwardX = (float)(Math.sin(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle)));
111         float forwardY = (float)(Math.cos(Math.toRadians(alphaViewAngle)) * Math.cos(Math.toRadians(betaViewAngle)));
112         float forwardZ = (float)(Math.sin(Math.toRadians(betaViewAngle)));
113
114         camX = camXPrev + dy * moveSpeed * forwardX;
115         camY = camYPrev + dy * moveSpeed * forwardY;
```


tivity.java
MyGLSurfaceView.java
MyGLRenderer.java
myWorkMode.java
myPyramidRotationMode.ja
5 public class myNineCubesMode extends myWorkMode {
105 public boolean onActionMove(float x, float y, int cx, int cy) {
116 camZ = camZPrev + dy * moveSpeed * forwardZ;
117
118 // Mouse sağ-sol hareket → bakış yönünü değiştir
119 alphaViewAngle = alphaAnglePrev + dx * 0.01f; //hız!
120
121 notifyCameraInfo();
122 return true;
123 }
124
2 usages
125 public void adjustAlpha(float delta) {
126 alphaViewAngle += delta;
127 notifyCameraInfo();
128 }
129
2 usages
130 public void adjustBeta(float delta) {
131 betaViewAngle += delta;
132 if (betaViewAngle > 89) betaViewAngle = 89;
133 if (betaViewAngle < -89) betaViewAngle = -89;
134 notifyCameraInfo();
135 }
136
1 usage
137 public float getAlpha() { return alphaViewAngle; }
1 usage
138 public float getBeta() { return betaViewAngle; }
1 usage
139 public float getCamX() { return camX; }
1 usage
140 public float getCamY() { return camY; }
1 usage
141 public float getCamZ() { return camZ; }

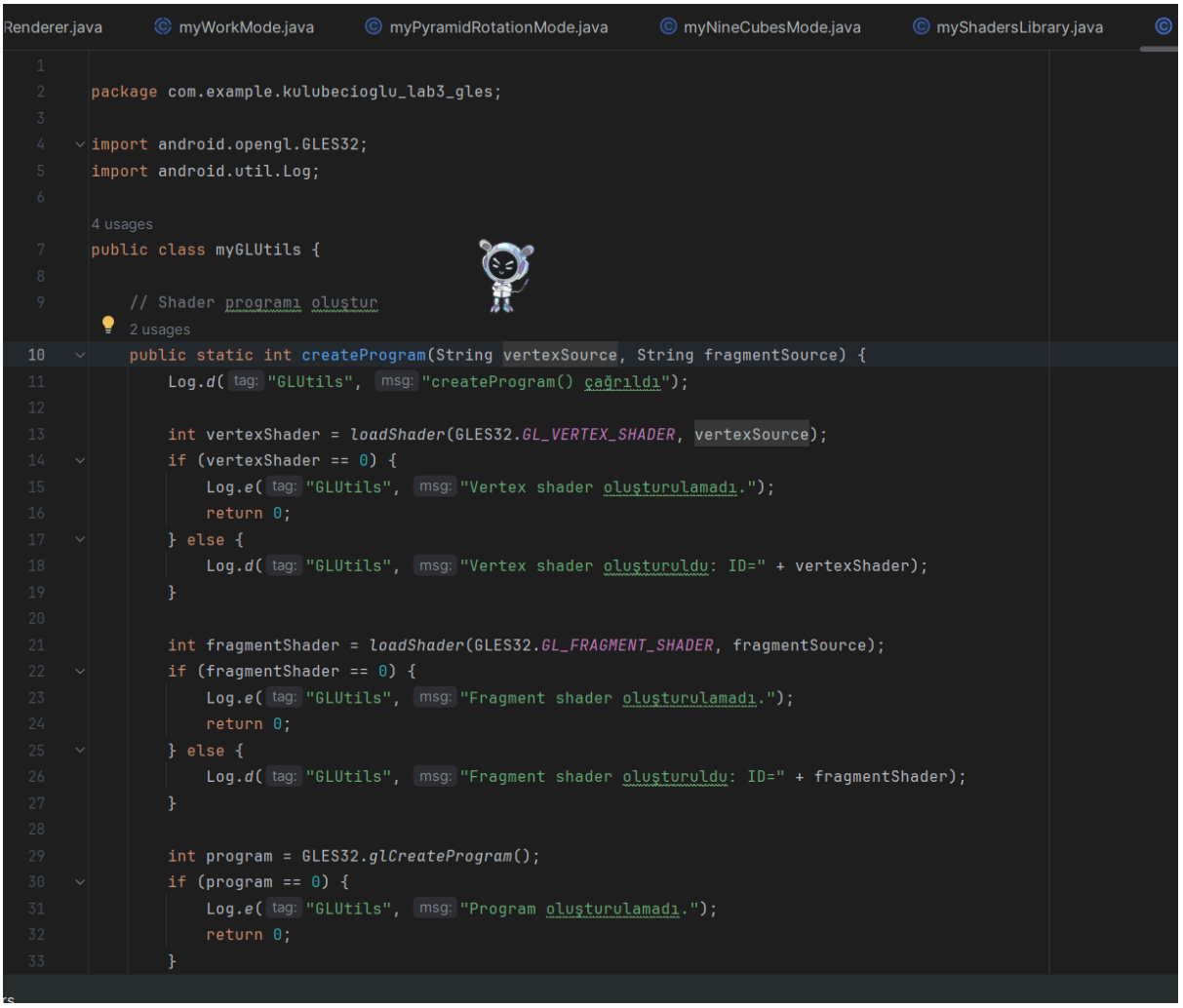
- **myShadersLibrary.java**

❖ Contains Vertex and Fragment shader source code.
❖ Used during shader program creation.



- **myGLUtils.java**

❖ Responsible for compiling shaders, creating VAOs, and binding vertex buffers.



```
            Log.e( tag: "GLUtils",  msg: "Program oluşturulamadı.");
            return 0;
        }

        GLES32.glAttachShader(program, vertexShader);
        GLES32.glAttachShader(program, fragmentShader);
        GLES32.glLinkProgram(program);

        int[] linkStatus = new int[1];
        GLES32.glGetProgramiv(program, GLES32.GL_LINK_STATUS, linkStatus,  offset: 0);
        if (linkStatus[0] != GLES32.GL_TRUE) {
            Log.e( tag: "GLUtils",  msg: "Program link hatası: " + GLES32.glGetProgramInfoLog(program));
            GLES32.glDeleteProgram(program);
            return 0;
        }

        Log.d( tag: "GLUtils",  msg: "Shader program başarıyla oluşturuldu. ID=" + program);

        GLES32.glDeleteShader(vertexShader);
        GLES32.glDeleteShader(fragmentShader);

        return program;
    }

    // Shader yükle
    2 usages
    public static int loadShader(int type, String shaderSource) {
        String shaderType = (type == GLES32.GL_VERTEX_SHADER) ? "VERTEX" : "FRAGMENT";
        Log.d( tag: "GLUtils",  msg: shaderType + " shader yükleniyor...");

        int shader = GLES32.glCreateShader(type);
        GLES32.glShaderSource(shader, shaderSource);
        GLES32.glCompileShader(shader);
```

```
public class myGLUtils {
    public static int loadShader(int type, String shaderSource) {

        int[] compiled = new int[1];
        GLES32.glGetShaderiv(shader, GLES32.GL_COMPILE_STATUS, compiled,  offset: 0);
        if (compiled[0] == 0) {
            Log.e( tag: "GLUtils",  msg: shaderType + " shader compile hatası: " + GLES32.glGetShaderInfoLog(shader));
            GLES32.glDeleteShader(shader);
            return 0;
        }

        Log.d( tag: "GLUtils",  msg: shaderType + " shader başarıyla derlendi. ID=" + shader);
        return shader;
    }

    // Vertex verisini GPU'ya bağla
    2 usages
    public static void bindVertexArrayWithColor(float[] arrayVertex, int glProgram) {
        Log.d( tag: "GLUtils",  msg: "Vertex buffer bind ediliyor...");
        java.nio.ByteBuffer bb = java.nio.ByteBuffer.allocateDirect( capacity: arrayVertex.length * 4);
        bb.order(java.nio.ByteOrder.nativeOrder());
        java.nio.FloatBuffer vertexBuffer = bb.asFloatBuffer();
        vertexBuffer.put(arrayVertex);
        vertexBuffer.position( newPosition: 0);

        int[] vbo = new int[1];
        GLES32.glGenBuffers( n: 1, vbo,  offset: 0);
        GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER, vbo[0]);
        GLES32.glBufferData(GLES32.GL_ARRAY_BUFFER,  size: arrayVertex.length * 4, vertexBuffer, GLES32.GL_STATIC_DRAW);

        int positionHandle = GLES32.glGetAttribLocation(glProgram,  name: "vPosition");
        int colorHandle = GLES32.glGetAttribLocation(glProgram,  name: "vColor");

        Log.d( tag: "GLUtils",  msg: "positionHandle: " + positionHandle + ", colorHandle: " + colorHandle);
```

```
94
95          if (positionHandle >= 0) {
96              GLES32.glEnableVertexAttribArray(positionHandle);
97              GLES32.glVertexAttribPointer(positionHandle, size: 3, GLES32.GL_FLOAT, normalized: false, stride: 6 * 4, offset: 0);
98          } else {
99              Log.e( tag: "GLUtils", msg: "vPosition attrib bulunamadı!");
100         }
101
102         if (colorHandle >= 0) {
103             GLES32.glEnableVertexAttribArray(colorHandle);
104             GLES32.glVertexAttribPointer(colorHandle, size: 3, GLES32.GL_FLOAT, normalized: false, stride: 6 * 4, offset: 3 * 4);
105         } else {
106             Log.e( tag: "GLUtils", msg: "vColor attrib bulunamadı!");
107         }
108
109         GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER, buffer: 0);
110         Log.d( tag: "GLUtils", msg: "Vertex buffer bind tamamlandı.");
111     }
112 }
113
```

4. Scene Features and User Interactions

Scene Name	Features
Pyramid Rotation	Pyramid rotates automatically, touch gestures rotate the scene or zoom
Nine Cubes	Free camera movement, navigation via touch and keyboard

- ❖ Both scenes show dynamic camera coordinate information (a, b, x, y, z) at the top of the screen.
- ❖ **`setCameraUpdateListener()`** is used for the pyramid scene.
- ❖ **`updateCameraInfoText()`** is used for the cube scene.

- **5. User Controls**

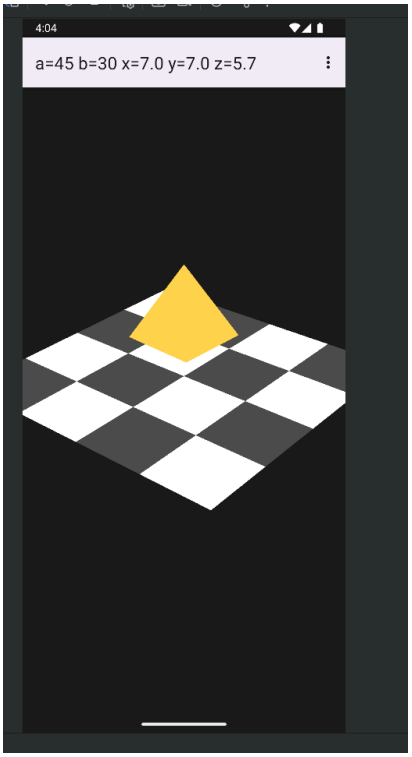
Input Method	Scene Effect
Touch (up/down swipe)	Zoom camera in/out
Touch (left/right swipe)	Rotate scene manually in pyramid mode / move in cube mode
Keyboard Arrow Keys	Adjust camera look direction in cube scene

- **6. Achievements**

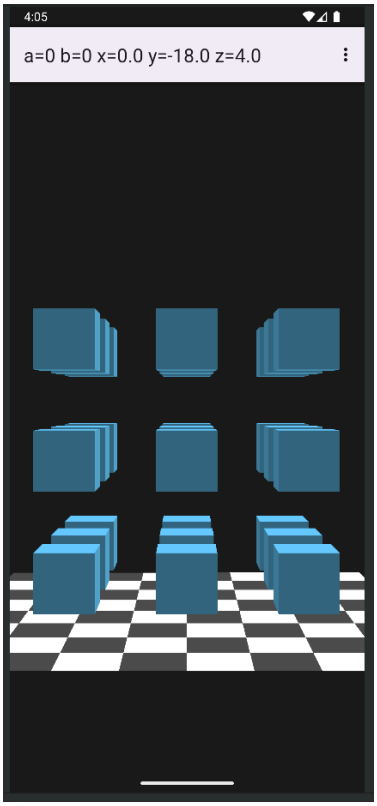
❖ 3D scene creation using MVP matrix
❖ Shader programming practice
❖ Implementation of free camera movement
❖ Real-time camera coordinate tracking
❖ Modular and extendable scene architecture

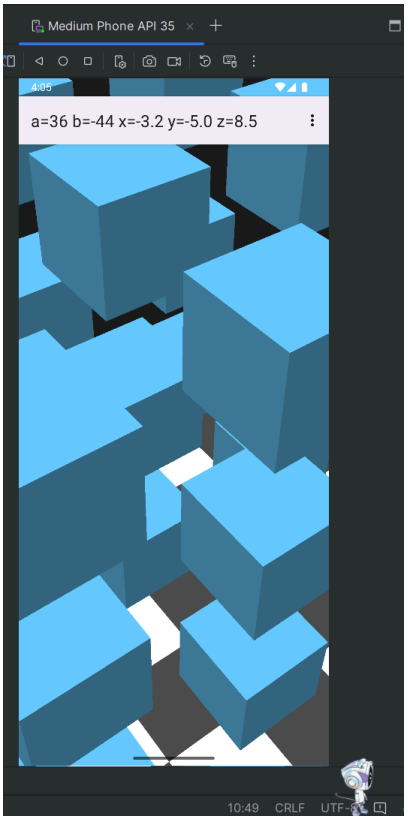
- **7. Screenshots**

❖ Pyramid Scene View

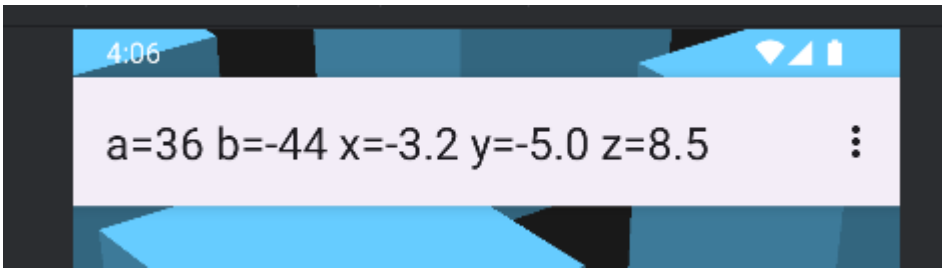


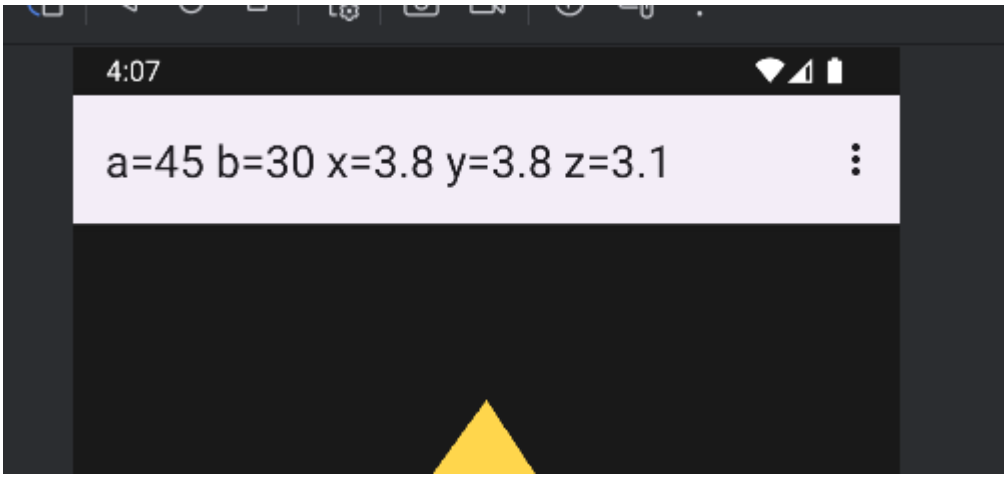
❖ Nine Cubes Scene View





❖ Dynamic camera coordinate display





- **8. Conclusion**

Through this lab project, we gained essential experience in OpenGL ES scene creation, shader programming, and user interaction integration. The modular design allows for scalability, enabling easy integration of new scenes and shapes. The project demonstrates successful implementation of real-time rendering, UI data synchronization, and interactive control logic.

Control Questions and Answers

❖ **What is a view coordinate system?**

The view coordinate system, also called the camera coordinate system or eye space, is a reference frame in which the scene is viewed from the camera's perspective. It is created by applying a view transformation (typically with a **`lookAt`** matrix or a combination of translations and rotations) to transform all objects from world space into a coordinate system relative to the camera. In this system, the camera is considered to be at the origin (0,0,0) looking toward a specific direction (usually the -Z axis).

❖ **How are coordinate transformations performed in the vertex shader?**

Coordinate transformations in the vertex shader are done by multiplying the vertex position by a series of transformation matrices:

- ➢ Model Matrix **(`uModelMatrix`)**: Transforms the object from local coordinates to world coordinates.
- ➢ View Matrix **(`uViewMatrix`)**: Transforms world coordinates into view (camera) coordinates.
- ➢ Projection Matrix **(`uProjMatrix`)**: Projects the 3D coordinates into 2D screen coordinates.

The final position is calculated as:

```
gl_Position = uProjMatrix * uViewMatrix * uModelMatrix * vec4(vPosition, 1.0);
```

❖ **How is the scene display angle determined and how to change it?**

The scene display angle is determined by the view transformation, particularly the camera's rotation angles (e.g., **`alphaViewAngle`, `betaViewAngle`**). These angles define how the scene is rotated around different axes. To change the view angle dynamically, the application often uses touch input events or variables that store the previous angle and modify it incrementally during interaction (e.g., dragging the finger on the screen updates the rotation angles).

❖ **How to specify the boundaries of the cone of visibility for perspective projection?**

The boundaries of the cone of visibility in a perspective projection are set using parameters in the **`Matrix.perspectiveM()`** function in Android OpenGL ES. The parameters include:

- ➢ field of view angle (fovy) – vertical angle of the cone.
- ➢ aspect ratio – width divided by height of the output screen.
- ➢ near and far planes – define the depth range of visible objects.

Example:

```
Matrix.perspectiveM(projMatrix, 0, 45f,
aspect, 0.1f, 30f);
```

- **How to take into account the ratio of the output window sizes to compensate for distortions in the proportions of the shape of objects?**
  This is done by calculating and applying the aspect ratio in the projection matrix. The aspect ratio is defined as the width divided by height of the rendering surface. It is passed as a parameter to the perspective matrix function: `float aspect = (float) width / height;` `Matrix.perspectiveM(projMatrix, 0, 45f, aspect, 0.1f, 30f);`
  This ensures that the shapes are not stretched or squashed when rendered.

- **How to determine the vertical angle of the camera cone of visibility?**
  The vertical angle of the camera's field of view (FOV) is defined in the `perspectiveM()` function as the first parameter (e.g., 45 degrees). It determines how "wide" the camera's viewing cone is. A larger angle shows more of the scene but can introduce distortion, while a smaller angle focuses more narrowly on objects.

- **How to program some actions for moving the stylus (finger) on the screen?**
  Actions for stylus (or finger) movement are handled using touch event callbacks in Android:

  - `onActionDown(float x, float y, int cx, int cy)` is triggered when the screen is touched.
  - `onActionMove(float x, float y, int cx, int cy)` is triggered when the finger moves on the screen. These functions allow the application to update camera angles (`alphaViewAngle`, `betaViewAngle`) or other interactive elements based on user input. For example, horizontal movement

might change the rotation around the Y-axis, and vertical movement might zoom in or out.