SPRAWOZDANIE

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium 3

Data 05.04.2022

Temat: "Modelowanie hierarchiczne w grafice 2D"

Wariant: Java 6-kąt

Patryk Lasik
Informatyka I stopień,
Zaoczne,
4 semestr,
Gr. Lab 2

1. **Polecenie:** Opracować scenę hierarchiczną zgodnie z obrazem używając zamiast kół wielokąty obracające się (animacja!) według wariantu. Opracowanie powinno być w jednym z języków: Java lub JavaScript,

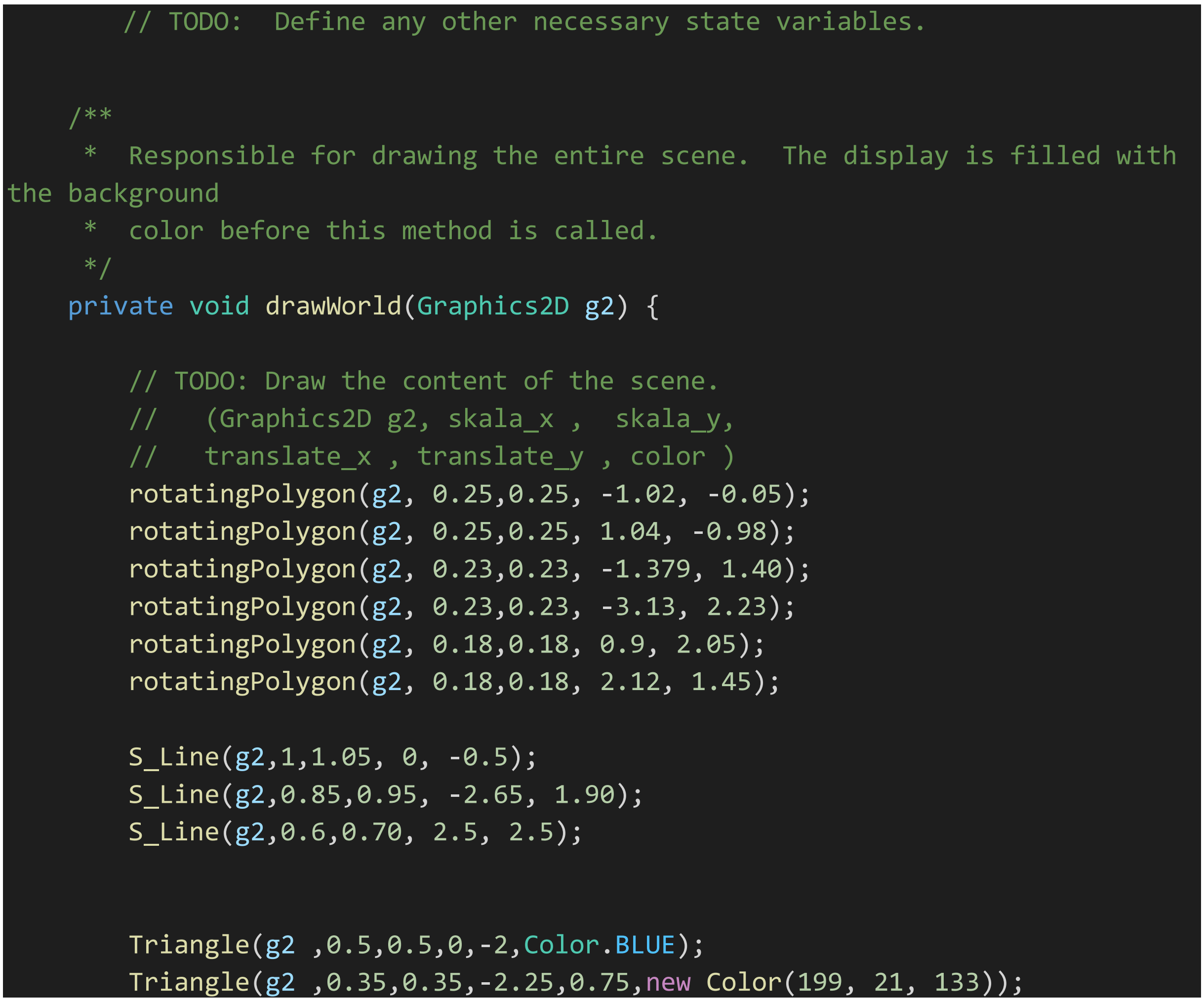
   na dwa sposoby:

   (a) używając hierarchię funkcje (sposób subroutinowy)

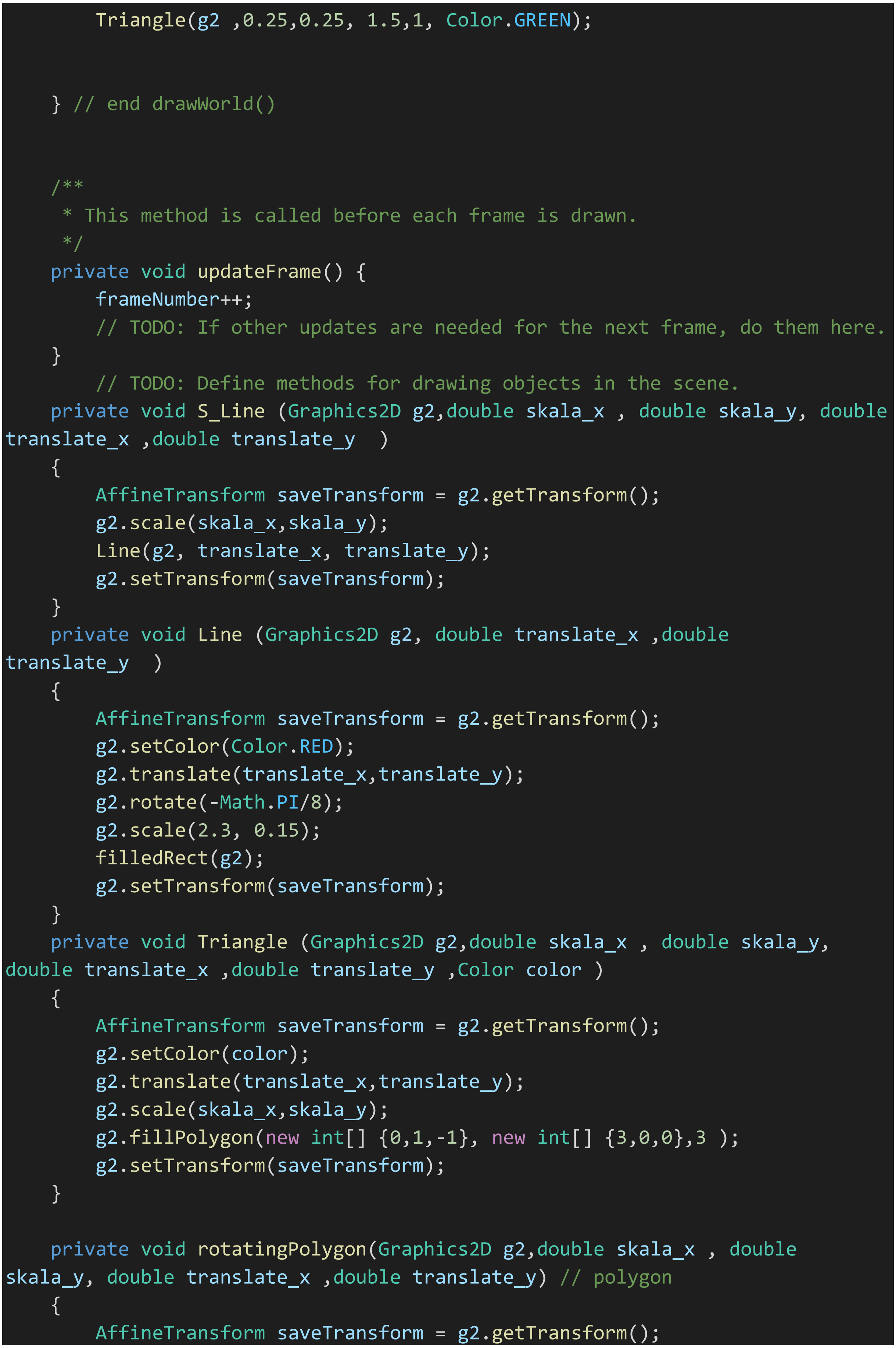
   (b) tworząc graf sceny (sposób obiektowy).
2. **Wprowadzane dane:**
   W pierwszym jak i zarówno drugim zadaniu użyłem różnych funkcji.
3. **Wykorzystane komendy:**

Wrzucam tutaj kod który napisałem/odpowiednio przekształciłem, całość znajduje się na moim GitHubie(link do niego podam poniżej)

Zadanie 1 Kod Źródłowy



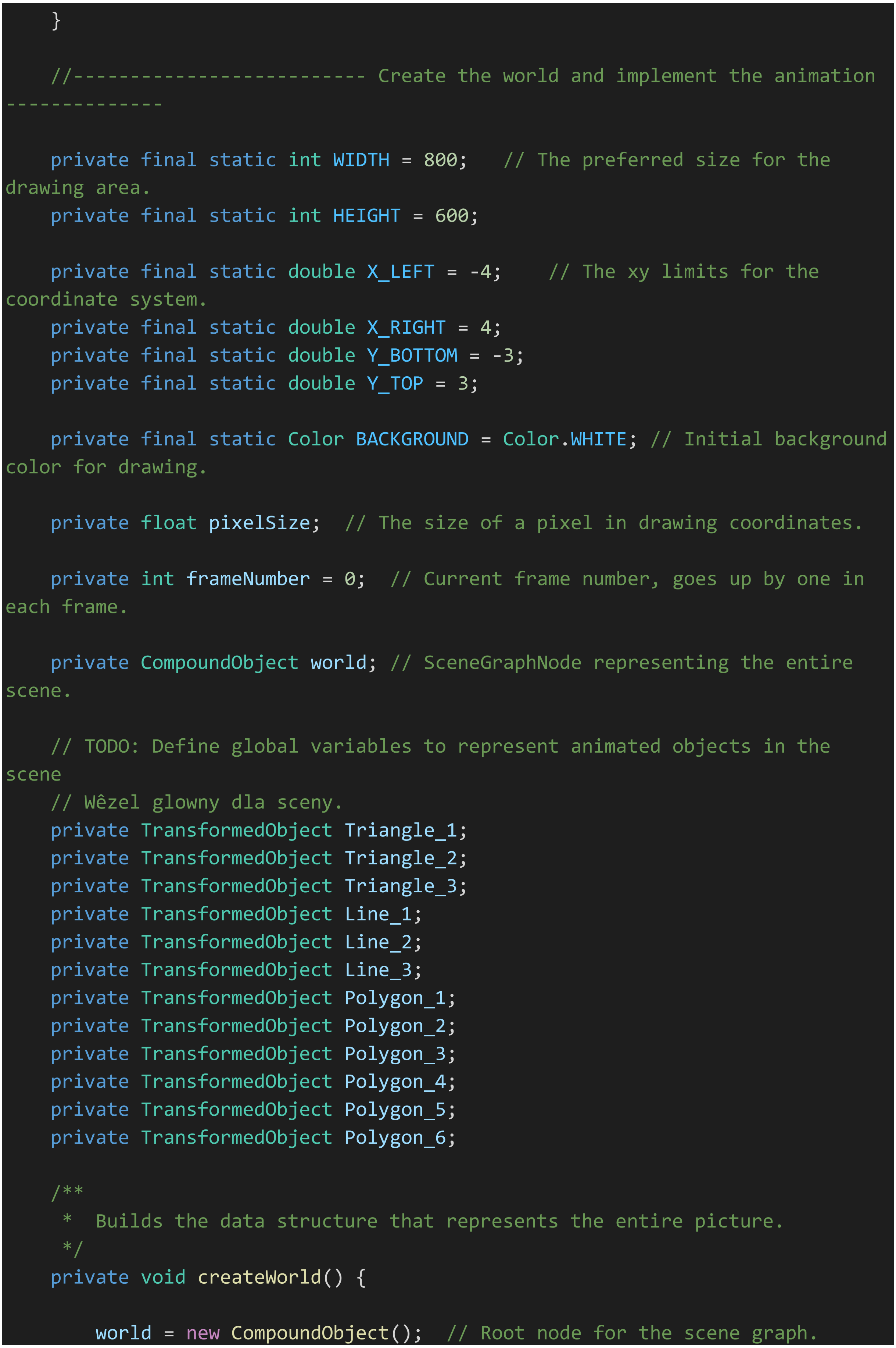
```
        // TODO:  Define any other necessary state variables.


    /**
     *  Responsible for drawing the entire scene.  The display is filled with
the background
     *  color before this method is called.
     */
    private void drawWorld(Graphics2D g2) {

        // TODO: Draw the content of the scene.
        //   (Graphics2D g2, skala_x ,  skala_y,
        //   translate_x , translate_y , color )
        rotatingPolygon(g2, 0.25,0.25, -1.02, -0.05);
        rotatingPolygon(g2, 0.25,0.25, 1.04, -0.98);
        rotatingPolygon(g2, 0.23,0.23, -1.379, 1.40);
        rotatingPolygon(g2, 0.23,0.23, -3.13, 2.23);
        rotatingPolygon(g2, 0.18,0.18, 0.9, 2.05);
        rotatingPolygon(g2, 0.18,0.18, 2.12, 1.45);

        S_Line(g2,1,1.05, 0, -0.5);
        S_Line(g2,0.85,0.95, -2.65, 1.90);
        S_Line(g2,0.6,0.70, 2.5, 2.5);


        Triangle(g2 ,0.5,0.5,0,-2,Color.BLUE);
        Triangle(g2 ,0.35,0.35,-2.25,0.75,new Color(199, 21, 133));
```



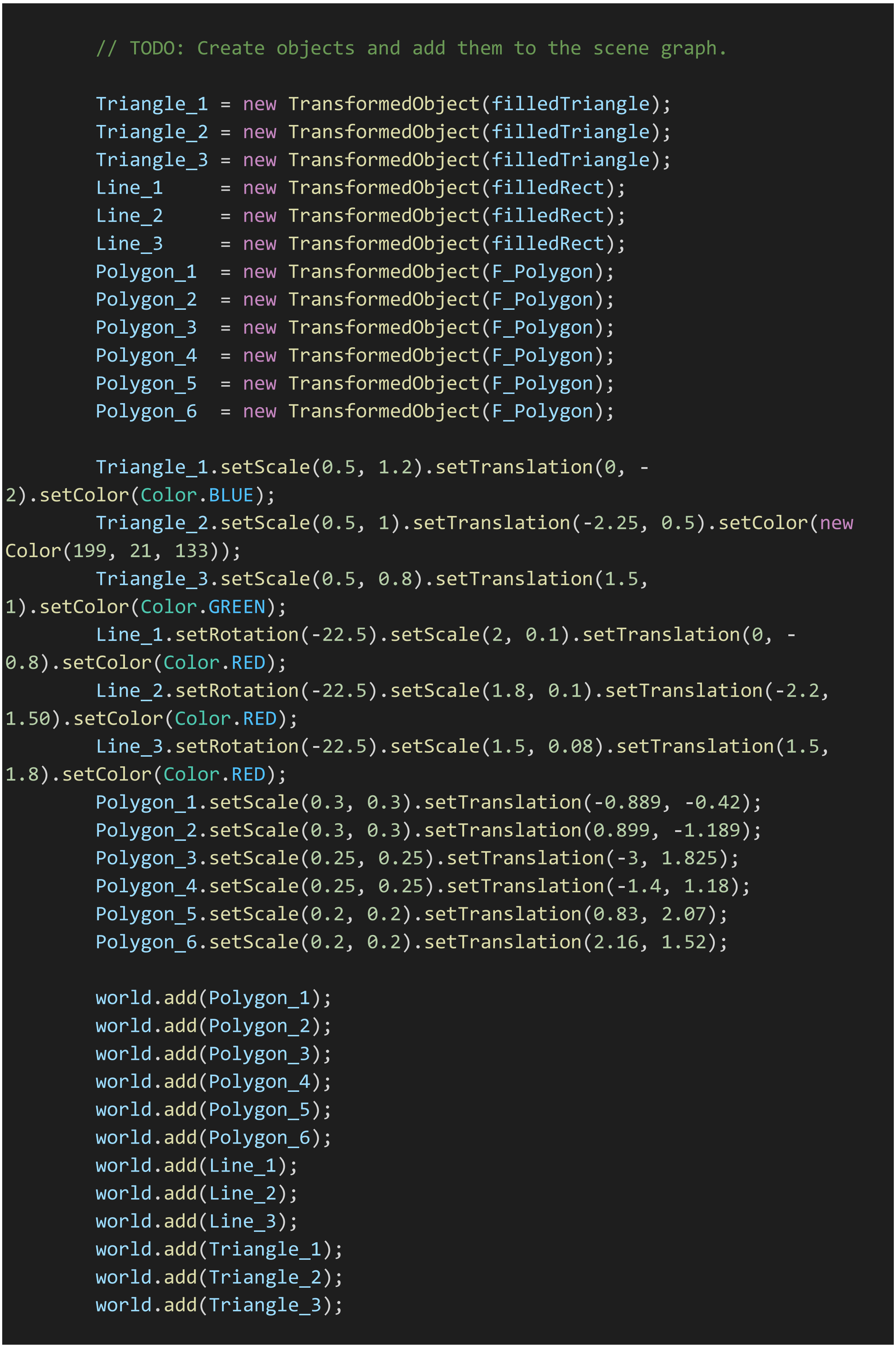
```
        Triangle(g2 ,0.25,0.25, 1.5,1, Color.GREEN);


    } // end drawWorld()


    /**
     * This method is called before each frame is drawn.
     */
    private void updateFrame() {
        frameNumber++;
        // TODO: If other updates are needed for the next frame, do them here.
    }
        // TODO: Define methods for drawing objects in the scene.
    private void S_Line (Graphics2D g2,double skala_x , double skala_y, double
translate_x ,double translate_y   )
    {
        AffineTransform saveTransform = g2.getTransform();
        g2.scale(skala_x,skala_y);
        Line(g2, translate_x, translate_y);
        g2.setTransform(saveTransform);
    }
    private void Line (Graphics2D g2, double translate_x ,double
translate_y   )
    {
        AffineTransform saveTransform = g2.getTransform();
        g2.setColor(Color.RED);
        g2.translate(translate_x,translate_y);
        g2.rotate(-Math.PI/8);
        g2.scale(2.3, 0.15);
        filledRect(g2);
        g2.setTransform(saveTransform);
    }
    private void Triangle (Graphics2D g2,double skala_x , double skala_y,
double translate_x ,double translate_y ,Color color )
    {
        AffineTransform saveTransform = g2.getTransform();
        g2.setColor(color);
        g2.translate(translate_x,translate_y);
        g2.scale(skala_x,skala_y);
        g2.fillPolygon(new int[] {0,1,-1}, new int[] {3,0,0},3 );
        g2.setTransform(saveTransform);
    }

    private void rotatingPolygon(Graphics2D g2,double skala_x , double
skala_y, double translate_x ,double translate_y) // polygon
    {
        AffineTransform saveTransform = g2.getTransform();
        Color saveColor = g2.getColor();
        g2.setColor( Color.BLACK );
        g2.translate(translate_x,translate_y);
        g2.rotate( Math.toRadians( frameNumber*0.75 ));
        g2.scale( skala_x, skala_y );
        g2.drawLine( -1, 2 , 1, 2 );
        g2.drawLine(  1, 2 , 2, 0 );
        g2.drawLine(  2, 0 , 1,-2 );
        g2.drawLine(  1,-2 ,-1,-2 );
        g2.drawLine( -1,-2 ,-2, 0 );
        g2.drawLine( -2, 0 ,-1, 2 );
        //szprychy
        g2.drawLine( -1, 2 , 1,-2 );
        g2.drawLine(  1, 2 ,-1,-2 );
        g2.drawLine( -2, 0 , 2, 0 );

        g2.setColor(saveColor);
        g2.setTransform(saveTransform);

    }
```

Zadanie 2 Kod Źródłowy

```
package gk.lab3;


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.geom.*;
import java.util.ArrayList;

/**
 * A panel that displays a two-dimensional animation that is constructed
 * using a scene graph to implement hierarchical modeling.  There is a
 * checkbox that turns the animation on and off.
 */
public class SceneGraph extends JPanel {

    public static void main(String[] args) {
        JFrame window = new JFrame("Scene Graph 2D");
        window.setContentPane( new SceneGraph() );
        window.pack();
        window.setLocation(100,60);
        window.setResizable(false);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);
```

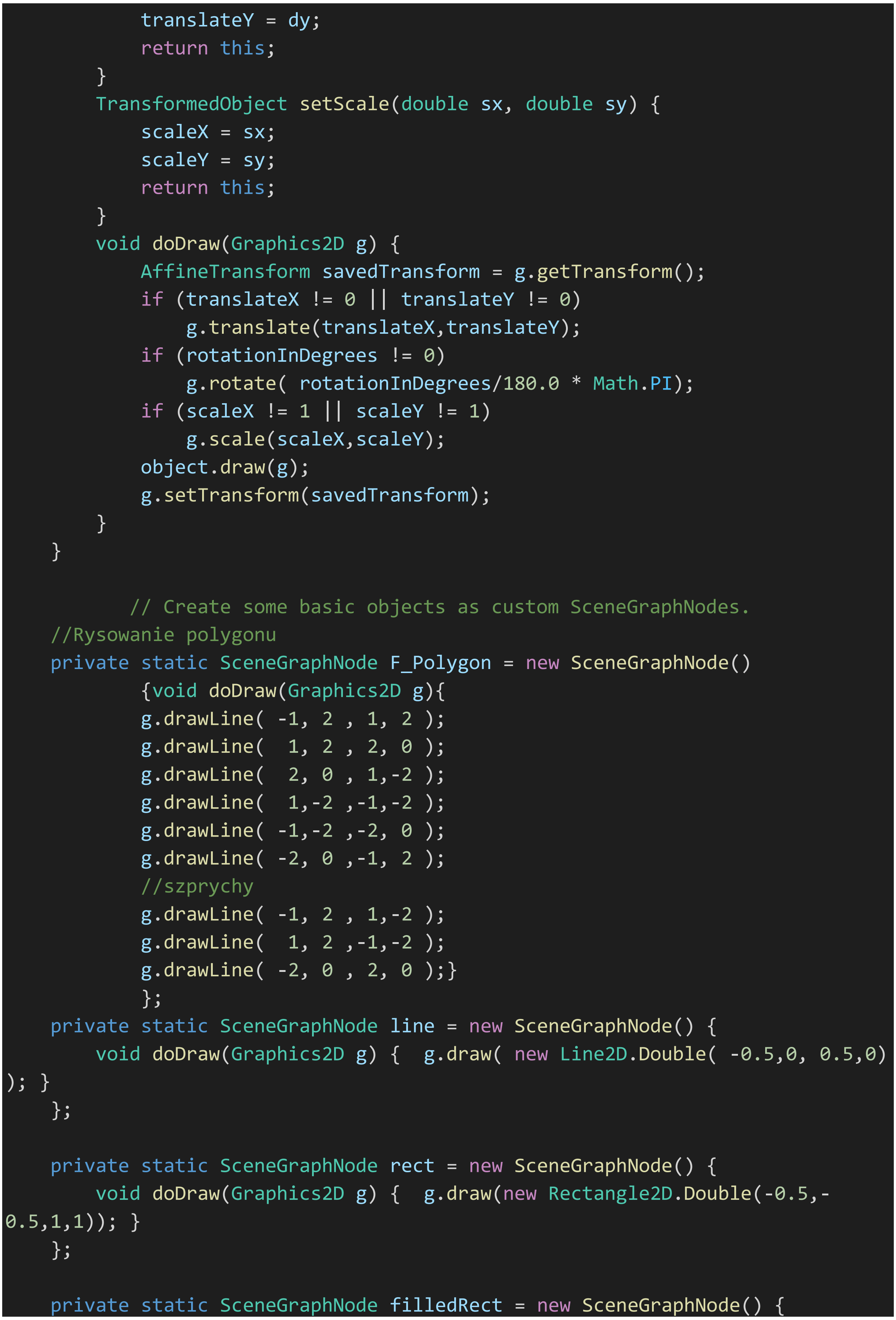


```
    }

    //------------------------- Create the world and implement the animation
--------------

    private final static int WIDTH = 800;   // The preferred size for the
drawing area.
    private final static int HEIGHT = 600;

    private final static double X_LEFT = -4;    // The xy limits for the
coordinate system.
    private final static double X_RIGHT = 4;
    private final static double Y_BOTTOM = -3;
    private final static double Y_TOP = 3;

    private final static Color BACKGROUND = Color.WHITE; // Initial background
color for drawing.

    private float pixelSize;  // The size of a pixel in drawing coordinates.

    private int frameNumber = 0;  // Current frame number, goes up by one in
each frame.

    private CompoundObject world; // SceneGraphNode representing the entire
scene.

    // TODO: Define global variables to represent animated objects in the
scene
    // Wêzel glowny dla sceny.
    private TransformedObject Triangle_1;
    private TransformedObject Triangle_2;
    private TransformedObject Triangle_3;
    private TransformedObject Line_1;
    private TransformedObject Line_2;
    private TransformedObject Line_3;
    private TransformedObject Polygon_1;
    private TransformedObject Polygon_2;
    private TransformedObject Polygon_3;
    private TransformedObject Polygon_4;
    private TransformedObject Polygon_5;
    private TransformedObject Polygon_6;

    /**
     *  Builds the data structure that represents the entire picture.
     */
    private void createWorld() {

        world = new CompoundObject();  // Root node for the scene graph.
```

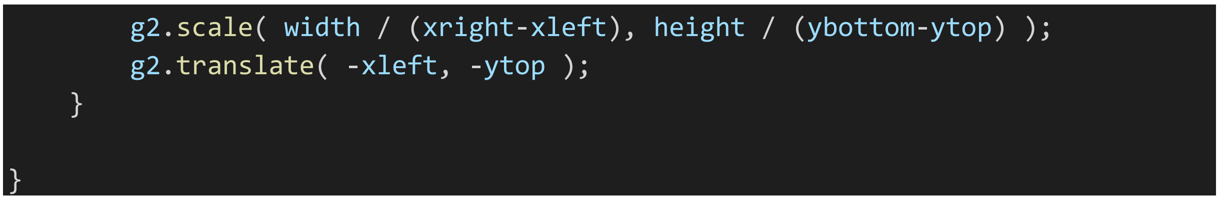


```
        // TODO: Create objects and add them to the scene graph.

        Triangle_1 = new TransformedObject(filledTriangle);
        Triangle_2 = new TransformedObject(filledTriangle);
        Triangle_3 = new TransformedObject(filledTriangle);
        Line_1     = new TransformedObject(filledRect);
        Line_2     = new TransformedObject(filledRect);
        Line_3     = new TransformedObject(filledRect);
        Polygon_1  = new TransformedObject(F_Polygon);
        Polygon_2  = new TransformedObject(F_Polygon);
        Polygon_3  = new TransformedObject(F_Polygon);
        Polygon_4  = new TransformedObject(F_Polygon);
        Polygon_5  = new TransformedObject(F_Polygon);
        Polygon_6  = new TransformedObject(F_Polygon);

        Triangle_1.setScale(0.5, 1.2).setTranslation(0, -
2).setColor(Color.BLUE);
        Triangle_2.setScale(0.5, 1).setTranslation(-2.25, 0.5).setColor(new
Color(199, 21, 133));
        Triangle_3.setScale(0.5, 0.8).setTranslation(1.5,
1).setColor(Color.GREEN);
        Line_1.setRotation(-22.5).setScale(2, 0.1).setTranslation(0, -
0.8).setColor(Color.RED);
        Line_2.setRotation(-22.5).setScale(1.8, 0.1).setTranslation(-2.2,
1.50).setColor(Color.RED);
        Line_3.setRotation(-22.5).setScale(1.5, 0.08).setTranslation(1.5,
1.8).setColor(Color.RED);
        Polygon_1.setScale(0.3, 0.3).setTranslation(-0.889, -0.42);
        Polygon_2.setScale(0.3, 0.3).setTranslation(0.899, -1.189);
        Polygon_3.setScale(0.25, 0.25).setTranslation(-3, 1.825);
        Polygon_4.setScale(0.25, 0.25).setTranslation(-1.4, 1.18);
        Polygon_5.setScale(0.2, 0.2).setTranslation(0.83, 2.07);
        Polygon_6.setScale(0.2, 0.2).setTranslation(2.16, 1.52);

        world.add(Polygon_1);
        world.add(Polygon_2);
        world.add(Polygon_3);
        world.add(Polygon_4);
        world.add(Polygon_5);
        world.add(Polygon_6);
        world.add(Line_1);
        world.add(Line_2);
        world.add(Line_3);
        world.add(Triangle_1);
        world.add(Triangle_2);
        world.add(Triangle_3);
```

```java
        //rotatingRect = new TransformedObject(filledRect);   // (DELETE THIS EXAMPLE)
        //rotatingRect.setScale(2,2).setColor(Color.RED);
        //world.add(rotatingRect);

    } // end createWorld()


    /**
     * This method is called just before each frame is drawn.  It updates the modeling
     * transformations of the objects in the scene that are animated.
     */
    public void updateFrame() {
        frameNumber++;

        // TODO: Update state in preparation for drawing the next frame.
        Polygon_1.setRotation(frameNumber*0.75);
        Polygon_2.setRotation(frameNumber*0.75);
        Polygon_3.setRotation(frameNumber*0.75);
        Polygon_4.setRotation(frameNumber*0.75);
        Polygon_5.setRotation(frameNumber*0.75);
        Polygon_6.setRotation(frameNumber*0.75);

    }



    //------------------- A Simple Scene Object-Oriented Scene Graph API ----------------

    private static abstract class SceneGraphNode {
        Color color;  // If not null, the default color for this node and its children.
                      // If null, the default color is inherited.
        SceneGraphNode setColor(Color c) {
            this.color = c;
            return this;
        }
        final void draw(Graphics2D g) {
            Color saveColor = null;
            if (color != null) {
                saveColor = g.getColor();
                g.setColor(color);
            }
            doDraw(g);
            if (saveColor != null) {
                g.setColor(saveColor);
            }
        }
        abstract void doDraw(Graphics2D g);
    }

    /**
     *  Defines a subclass, CompoundObject, of SceneGraphNode to represent
     *  an object that is made up of sub-objects.  Initially, there are no
     *  sub-objects.  Objects are added with the add() method.
     */
    private static class CompoundObject extends SceneGraphNode {
        ArrayList<SceneGraphNode> subobjects = new
ArrayList<SceneGraphNode>();
        CompoundObject add(SceneGraphNode node) {
            subobjects.add(node);
            return this;
        }
        void doDraw(Graphics2D g) {
            for (SceneGraphNode node : subobjects)
                node.draw(g);
        }
    }

    /**
     *  TransformedObject is a subclass of SceneGraphNode that
     *  represents an object along with a modeling transformation to
     *  be applied to that object.  The object must be specified in
     *  the constructor.  The transformation is specified by calling
     *  the setScale(), setRotate() and setTranslate() methods. Note that
     *  each of these methods returns a reference to the TransformedObject
     *  as its return value, to allow for chaining of method calls.
     *  The modeling transformations are always applied to the object
     *  in the order scale, then rotate, then translate.
     */
    private static class TransformedObject extends SceneGraphNode {
        SceneGraphNode object;
        double rotationInDegrees = 0;
        double scaleX = 1, scaleY = 1;
        double translateX = 0, translateY = 0;
        TransformedObject(SceneGraphNode object) {
            this.object = object;
        }
        TransformedObject setRotation(double degrees) {
            rotationInDegrees = degrees;
            return this;
        }
        TransformedObject setTranslation(double dx, double dy) {
            translateX = dx;
```



```
            translateY = dy;
            return this;
        }
        TransformedObject setScale(double sx, double sy) {
            scaleX = sx;
            scaleY = sy;
            return this;
        }
        void doDraw(Graphics2D g) {
            AffineTransform savedTransform = g.getTransform();
            if (translateX != 0 || translateY != 0)
                g.translate(translateX,translateY);
            if (rotationInDegrees != 0)
                g.rotate( rotationInDegrees/180.0 * Math.PI);
            if (scaleX != 1 || scaleY != 1)
                g.scale(scaleX,scaleY);
            object.draw(g);
            g.setTransform(savedTransform);
        }
    }

           // Create some basic objects as custom SceneGraphNodes.
    //Rysowanie polygonu
    private static SceneGraphNode F_Polygon = new SceneGraphNode()
            {void doDraw(Graphics2D g){
            g.drawLine( -1, 2 , 1, 2 );
            g.drawLine(  1, 2 , 2, 0 );
            g.drawLine(  2, 0 , 1,-2 );
            g.drawLine(  1,-2 ,-1,-2 );
            g.drawLine( -1,-2 ,-2, 0 );
            g.drawLine( -2, 0 ,-1, 2 );
            //szprychy
            g.drawLine( -1, 2 , 1,-2 );
            g.drawLine(  1, 2 ,-1,-2 );
            g.drawLine( -2, 0 , 2, 0 );}
            };
    private static SceneGraphNode line = new SceneGraphNode() {
        void doDraw(Graphics2D g) {  g.draw( new Line2D.Double( -0.5,0, 0.5,0)
); }
    };

    private static SceneGraphNode rect = new SceneGraphNode() {
        void doDraw(Graphics2D g) {  g.draw(new Rectangle2D.Double(-0.5,-
0.5,1,1)); }
    };

    private static SceneGraphNode filledRect = new SceneGraphNode() {
        void doDraw(Graphics2D g) {  g.fill(new Rectangle2D.Double(-0.5,-
0.5,1,1)); }
    };

    private static SceneGraphNode circle = new SceneGraphNode() {
        void doDraw(Graphics2D g) {  g.draw(new Ellipse2D.Double(-0.5,-
0.5,1,1)); }
    };

    private static SceneGraphNode filledCircle = new SceneGraphNode() {
        void doDraw(Graphics2D g) {  g.fill(new Ellipse2D.Double(-0.5,-
0.5,1,1)); }
    };

    private static SceneGraphNode filledTriangle = new SceneGraphNode() {
        void doDraw(Graphics2D g) {  // width = 1, height = 1, center of base
is at (0,0);
            Path2D path = new Path2D.Double();
            path.moveTo(-0.5,0);
            path.lineTo(0.5,0);
            path.lineTo(0,1);
            path.closePath();
            g.fill(path);
        }
    };



    //-------------------------------- Implementation ----------------------
-------------

    private JPanel display;  // The JPanel in which the scene is drawn.

    /**
     * Constructor creates the scene graph data structure that represents the
     * scene that is to be drawn in this panel, by calling createWorld().
     * It also sets the preferred size of the panel to the constants WIDTH and
HEIGHT.
     * And it creates a timer to drive the animation.
     */
    public SceneGraph() {
        display = new JPanel() {
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                Graphics2D g2 = (Graphics2D)g.create();
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
                applyLimits(g2, X_LEFT, X_RIGHT, Y_TOP, Y_BOTTOM, false);
                g2.setStroke( new BasicStroke(pixelSize) ); // set default line width to one pixel.
                world.draw(g2);
            }
        };
        display.setPreferredSize( new Dimension(WIDTH,HEIGHT));
        display.setBackground( BACKGROUND );
        final Timer timer = new Timer(17,new ActionListener() { // about 60 frames per second
            public void actionPerformed(ActionEvent evt) {
                updateFrame();
                repaint();
            }
        });
        final JCheckBox animationCheck = new JCheckBox("Run Animation");
        animationCheck.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                if (animationCheck.isSelected()) {
                    if ( ! timer.isRunning() )
                        timer.start();
                }
                else {
                    if ( timer.isRunning() )
                        timer.stop();
                }
            }
        });
        JPanel top = new JPanel();
        top.add(animationCheck);
        setLayout(new BorderLayout(5,5));
        setBackground(Color.DARK_GRAY);
        setBorder( BorderFactory.createLineBorder(Color.DARK_GRAY,4) );
        add(top,BorderLayout.NORTH);
        add(display,BorderLayout.CENTER);
        createWorld();
    }



    /**
     * Applies a coordinate transform to a Graphics2D graphics context.  The upper left corner of
     * the viewport where the graphics context draws is assumed to be (0,0).  The coordinate
     * transform will make a requested rectangle visible in the drawing area.  The requested
     * limits might be adjusted to preserve the aspect ratio.  (This method sets the global variable
     * pixelSize to be equal to the size of one pixel in the transformed
coordinate system.)
     * @param g2 The drawing context whose transform will be set.
     * @param xleft requested x-value at left of drawing area.
     * @param xright requested x-value at right of drawing area.
     * @param ytop requested y-value at top of drawing area.
     * @param ybottom requested y-value at bottom of drawing area; can be less
than ytop, which will
     *     reverse the orientation of the y-axis to make the positive
direction point upwards.
     * @param preserveAspect if preserveAspect is false, then the requested
rectangle will exactly fill
     * the viewport; if it is true, then the limits will be expanded in one
direction, horizontally or
     * vertically, to make the aspect ratio of the displayed rectangle match
the aspect ratio of the
     * viewport.  Note that when preserveAspect is false, the units of measure
in the horizontal and
     * vertical directions will be different.
     */
    private void applyLimits(Graphics2D g2, double xleft, double xright,
            double ytop, double ybottom, boolean preserveAspect) {
        int width = display.getWidth();   // The width of the drawing area, in
pixels.
        int height = display.getHeight(); // The height of the drawing area,
in pixels.
        if (preserveAspect) {
            // Adjust the limits to match the aspect ratio of the drawing
area.
            double displayAspect = Math.abs((double)height / width);
            double requestedAspect = Math.abs(( ybottom-ytop ) / ( xright-
xleft ));
            if (displayAspect > requestedAspect) {
                double excess = (ybottom-ytop) *
(displayAspect/requestedAspect - 1);
                ybottom += excess/2;
                ytop -= excess/2;
            }
            else if (displayAspect < requestedAspect) {
                double excess = (xright-xleft) *
(requestedAspect/displayAspect - 1);
                xright += excess/2;
                xleft -= excess/2;
            }
        }
        double pixelWidth = Math.abs(( xright - xleft ) / width);
        double pixelHeight = Math.abs(( ybottom - ytop ) / height);
        pixelSize = (float)Math.min(pixelWidth,pixelHeight);
```

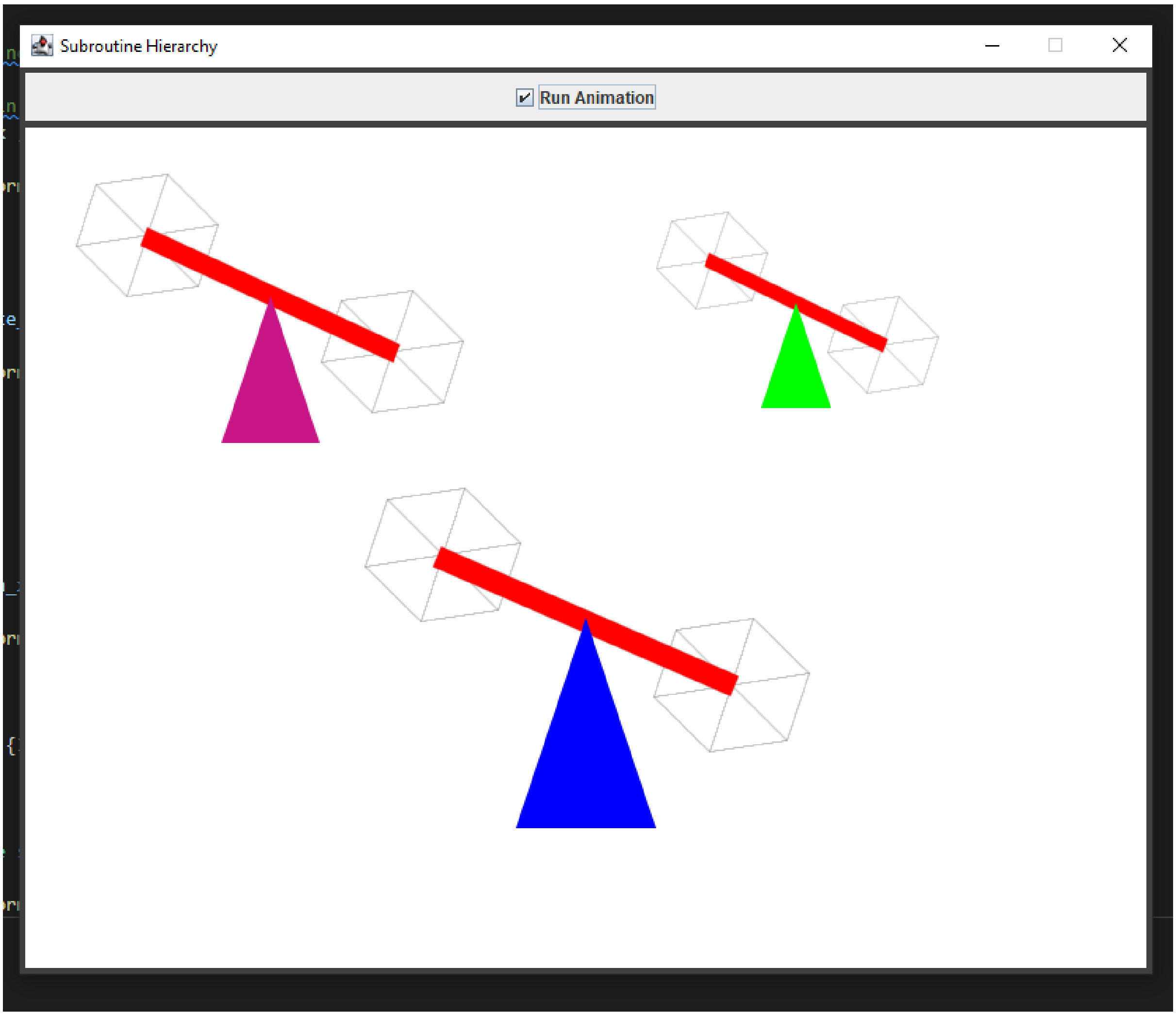


```
        g2.scale( width / (xright-xleft), height / (ybottom-ytop) );
        g2.translate( -xleft, -ytop );
    }

}
```

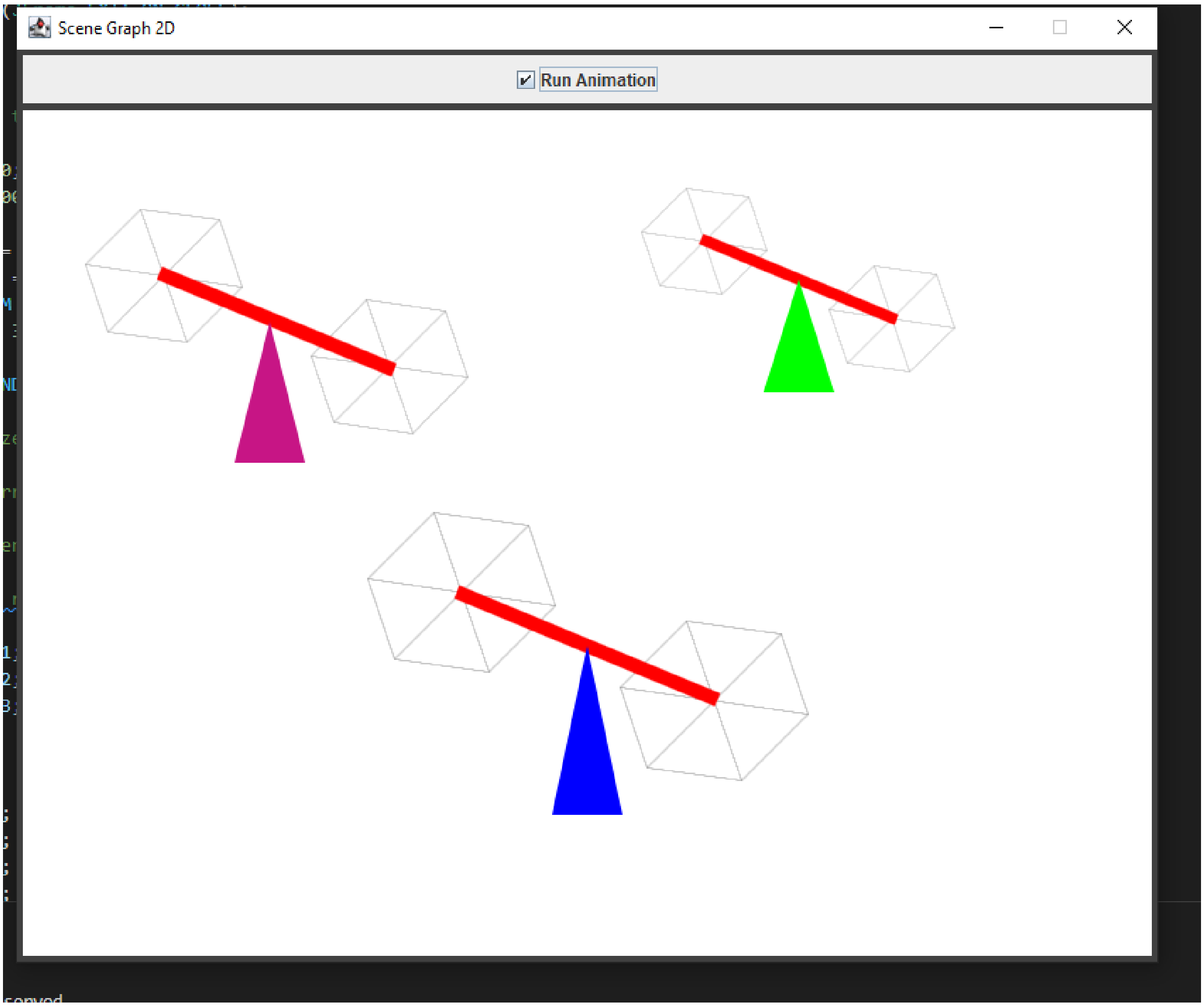
Link do zdalnego repozytorium (GitHub)
 https://github.com/LasikP/GrafikaKomputerowa

4. Wynik działania:

Zadanie 1:



Zadanie 2:



5. Wnioski:

Celem jest użycie Javy na dwa sposoby, pierwszy sposób subroutinowy a drugi sposób obiektowy. Zadanie pokazuje różnicę użycia tych dwóch sposobów ale wynik działania jest taki sam.