SPRAWOZDANIE
Zajęcia: Grafika komputerowa
Prowadzący: prof. dr. hab. Vasyl Martsenyuk

Laboratorium 3
11 III 2021 r.
Temat: "Modelowanie hierarchiczne w grafice 2D"
Liczba kątów:5

Robert Laszczak
Informatyka I stopień
Stacjonarne, 4 semestr
Grupa 2B

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 funkcję (sposób subroutinowy)
b) Tworząc graf sceny (sposób obiektowy)

2. Wprowadzone dane:

Liczba kątów n = 5

3. Wykorzystane komendy:

a) Kod źródłowy:

```
package lab3a
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.geom.*;
import java.util.Random;

/**
 * A panel that displays a two-dimensional animation that is drawn
 * using subroutines to implement hierarchical modeling.  There is a
 * checkbox that turns the animation on and off.
 */
public class Main extends JPanel {
   public static void main(String[] args) {
      JFrame window = new JFrame("Subroutine Hierarchy");
      window.setContentPane(new Main());
      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.

   // 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.
    */
   Random rand = new Random();
   float r,g,b;
   private void koloruj(){
      r=rand.nextFloat();
      g=rand.nextFloat();
      b=rand.nextFloat();
   }
   private void funcja(Graphics2D g2, double a, double tranX, double tranY) {
      AffineTransform basicTransform = g2.getTransform();
      g2.translate(0*a+tranX, -2*a+tranY);
```

```
    g2.scale(0.5*a, 1*a);
    koloruj();
    g2.setColor(new Color(r,g,b));
    filledTriangle(g2);
    g2.setTransform(basicTransform);
    g2.translate(0.65*a+tranX,-1.25*a+tranY);
    g2.scale(0.75*a,0.75*a);
    rotatingPolygon(g2);
    g2.setTransform(basicTransform);
    g2.translate(-0.65*a+tranX,-0.75*a+tranY);
    g2.scale(0.75*a,0.75*a);
    rotatingPolygon(g2);
    g2.setTransform(basicTransform);
    g2.translate(0*a+tranX, -1*a+tranY);
    g2.rotate(Math.toRadians(-20));
    g2.scale(1.5*a, 0.1*a);
    koloruj();
    g2.setColor(new Color(r,g,b));
    filledRect(g2);
    g2.setTransform(basicTransform);
}
private void drawWorld(Graphics2D g2) {
    funcja(g2,1.5,0,0.5);
    funcja(g2,1,-2,1.5);
    funcja(g2,0.75,2,2.5);
    /*
       AffineTransform basicTransform = g2.getTransform();
       g2.translate(0*a+tranX, -2*a+tranY);
       g2.scale(0.5*a, 1*a);
       koloruj();
       g2.setColor(new Color(r,g,b));
       filledTriangle(g2);
       g2.setTransform(basicTransform);
       g2.translate(0.65*a+tranX,-1.25*a+tranY);
       g2.scale(0.75*a,0.75*a);
       rotatingPolygon(g2);
       g2.setTransform(basicTransform);
       g2.translate(-0.65*a+tranX,-0.75*a+tranY);
       g2.scale(0.75*a,0.75*a);
       rotatingPolygon(g2);
       g2.setTransform(basicTransform);
       g2.translate(0*a+tranX, -1*a+tranY);
       g2.rotate(Math.toRadians(-20));
       g2.scale(1.5*a, 0.1*a);
       koloruj();
       g2.setColor(new Color(r,g,b));
       filledRect(g2);
       g2.setTransform(basicTransform);
     */

    // TODO: Draw the content of the scene.
} // 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 rotatingRect(Graphics2D g2) { // (DELETE THIS EXAMPLE)
    AffineTransform saveTransform = g2.getTransform();  // (It might be necessary to save/restore transform and color)
    Color saveColor = g2.getColor();
    g2.setColor(Color.RED);
    g2.rotate(Math.toRadians(frameNumber * 0.75));
    g2.scale(2, 2);
    filledRect(g2);
    g2.setColor(saveColor);
    g2.setTransform(saveTransform);
```

```
    }
    private void rotatingPolygon(Graphics2D g2) { // (DELETE THIS EXAMPLE)
        AffineTransform saveTransform = g2.getTransform();  // (It might be necessary to save/restore transform and color)
        Color saveColor = g2.getColor();
        g2.setColor(Color.BLACK);
        g2.rotate(Math.toRadians(frameNumber * 0.75));
        g2.scale(2, 2);
        filledPolygon(g2);
        g2.setColor(saveColor);
        g2.setTransform(saveTransform);
    }

    //------------------- Some methods for drawing basic shapes. ----------------
    private static void line(Graphics2D g2) { // Draws a line from (-0.5,0) to (0.5,0)
        g2.draw(new Line2D.Double(-0.5, 0, 0.5, 0));
    }
    private static void rect(Graphics2D g2) { // Strokes a square, size = 1, center = (0,0)
        g2.draw(new Rectangle2D.Double(-0.5, -0.5, 1, 1));
    }
    private static void filledRect(Graphics2D g2) { // Fills a square, size = 1, center = (0,0)
        g2.fill(new Rectangle2D.Double(-0.5, -0.5, 1, 1));
    }
    private static void circle(Graphics2D g2) { // Strokes a circle, diameter = 1, center = (0,0)
        g2.draw(new Ellipse2D.Double(-0.5, -0.5, 1, 1));
    }
    private static void filledCircle(Graphics2D g2) { // Fills a circle, diameter = 1, center = (0,0)
        g2.draw(new Ellipse2D.Double(-0.5, -0.5, 1, 1));
    }
    private static void filledTriangle(Graphics2D g2) { // 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();
        g2.fill(path);
    }
    private static void filledPolygon(Graphics2D g2) {
        int R = 50;
        int n = 5;
        int[] xPoints = new int[n];
        int[] yPoints = new int[n];
        for (int i = 0; i < n; i++) {
            xPoints[i] = (int) (R * Math.cos((2 * Math.PI * i) / n));
            yPoints[i] = (int) (R * Math.sin((2 * Math.PI * i) / n));
        }
        Polygon polygon = new Polygon(xPoints, yPoints, n);
        g2.translate(0, 0);
        g2.scale(0.003, 0.003); //skalowanie
        g2.setStroke(new BasicStroke(5));
        //wypeĹ‚nienie
        g2.drawPolygon(polygon);    //narysowanie
        g2.setStroke(new BasicStroke(2));
        for (int i = 0; i < n; i++) {
            g2.drawLine(0, 0, xPoints[i], yPoints[i]);
        }
    }
    private static void resetTransform(Graphics2D g2) {
        g2.setTransform(new AffineTransform());
    }

//---------------------------------- 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 Main() {
        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.
         drawWorld(g2);  // draw the world
      }

   };
   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);
}

/**
 * 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);
   }
}
```

b) Kod źródłowy:

```
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 Main extends JPanel {
   public static void main(String[] args) {
      JFrame window = new JFrame("Scene Graph 2D");
      window.setContentPane( new Main() );
      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.
   private TransformedObject rotatingRect;  // (DELETE THIS EXAMPLE)
   private TransformedObject rotatingPolygon1;
```

```
    private TransformedObject rotatingPolygon2;
    private TransformedObject rotatingPolygon3;
    private TransformedObject rotatingPolygon4;
    private TransformedObject rotatingPolygon5;
    private TransformedObject rotatingPolygon6;

    private TransformedObject staticRect1;
    private TransformedObject staticRect2;
    private TransformedObject staticRect3;

    private TransformedObject staticTriangle1;
    private TransformedObject staticTriangle2;
    private TransformedObject staticTriangle3;

    /**
     *  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.
        rotatingPolygon1 = new TransformedObject(filledPolygon);   // (DELETE THIS EXAMPLE)
        rotatingPolygon1.setScale(2,2).setColor(Color.BLACK); //lewy lewe kolo
        rotatingPolygon1.setTranslation(-2.5,2);
        world.add(rotatingPolygon1);

        rotatingPolygon2 = new TransformedObject(filledPolygon);
        rotatingPolygon2.setTranslation(-0.9,1.4);
        rotatingPolygon2.setScale(2,2).setColor(Color.BLACK); //lewy prawe kolo
        world.add(rotatingPolygon2);

        staticRect1 =new TransformedObject(filledRect); //lewy ramie
        staticRect1.setTranslation(-1.7,1.7);
        staticRect1.setScale(2,0.1);
        staticRect1.setRotation(-20);

        staticRect1.setColor(Color.GRAY); //lewy ramiÄ™
        world.add(staticRect1);

        staticTriangle1 = new TransformedObject(filledTriangle); //lewy trojkat
        staticTriangle1.setTranslation(-1.7,0.5);
        staticTriangle1.setScale(0.8,1.2);
        staticTriangle1.setColor(Color.BLACK);
        world.add(staticTriangle1);

    //1Spining
        double a =1.5;

        rotatingPolygon3 = new TransformedObject(filledPolygon);   // (DELETE THIS EXAMPLE)
        rotatingPolygon3.setScale(2*a,2*a).setColor(Color.BLACK); //dol lewe kolo
        rotatingPolygon3.setTranslation(-0.8,-0.2);
        world.add(rotatingPolygon3);

        rotatingPolygon4 = new TransformedObject(filledPolygon);
        rotatingPolygon4.setTranslation(1.8, -1.2);
        rotatingPolygon4.setScale(2*a,2*a).setColor(Color.BLACK); //dol prawe kolo
        world.add(rotatingPolygon4);

        staticRect2 =new TransformedObject(filledRect); //dol ramie
        staticRect2.setTranslation(0.5,-0.7);
        staticRect2.setScale(2*a,0.1*a);
        staticRect2.setRotation(-20);

        staticRect2.setColor(Color.BLUE); //dol, ramie
        world.add(staticRect2);

        staticTriangle2 = new TransformedObject(filledTriangle); //dol, trojkat
        staticTriangle2.setTranslation(0.5,-2.5);
        staticTriangle2.setScale(0.8*a,1.2*a);
        staticTriangle2.setColor(Color.YELLOW);
        world.add(staticTriangle2);
```

```
   //1Spining
   a =0.7;

   rotatingPolygon6 = new TransformedObject(filledPolygon);   // (DELETE THIS EXAMPLE)
   rotatingPolygon6.setScale(2*a,2*a).setColor(Color.BLACK); //prawy prawe kolo
   rotatingPolygon6.setTranslation(2.1,1.6);
   world.add(rotatingPolygon6);

   rotatingPolygon5 = new TransformedObject(filledPolygon);
   rotatingPolygon5.setTranslation(0.9,2.02);
   rotatingPolygon5.setScale(2*a,2*a).setColor(Color.BLACK); //prawy lewe kolo
   world.add(rotatingPolygon5);

   staticRect3 =new TransformedObject(filledRect); //prawy ramie™
   staticRect3.setTranslation(1.5,1.8);
   staticRect3.setScale(2*a,0.1*a);
   staticRect3.setRotation(-20);

   staticRect3.setColor(Color.ORANGE); //prawy ramie
   world.add(staticRect3);

   staticTriangle3 = new TransformedObject(filledTriangle); //prawy trojkat
   staticTriangle3.setTranslation(1.5,1);
   staticTriangle3.setScale(0.8*a,1.2*a);
   staticTriangle3.setColor(Color.GREEN);
   world.add(staticTriangle3);

} // 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.

   rotatingPolygon1.setRotation(frameNumber*0.75); // (DELETE THIS EXAMPLE)
   rotatingPolygon2.setRotation(frameNumber*0.75); // (DELETE THIS EXAMPLE)
   rotatingPolygon3.setRotation(frameNumber*0.75); // (DELETE THIS EXAMPLE)
   rotatingPolygon4.setRotation(frameNumber*0.75); // (DELETE THIS EXAMPLE)
   rotatingPolygon5.setRotation(frameNumber*0.75); // (DELETE THIS EXAMPLE)
   rotatingPolygon6.setRotation(frameNumber*0.75); // (DELETE THIS EXAMPLE)
}

//------------------- 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);
}

/**
```

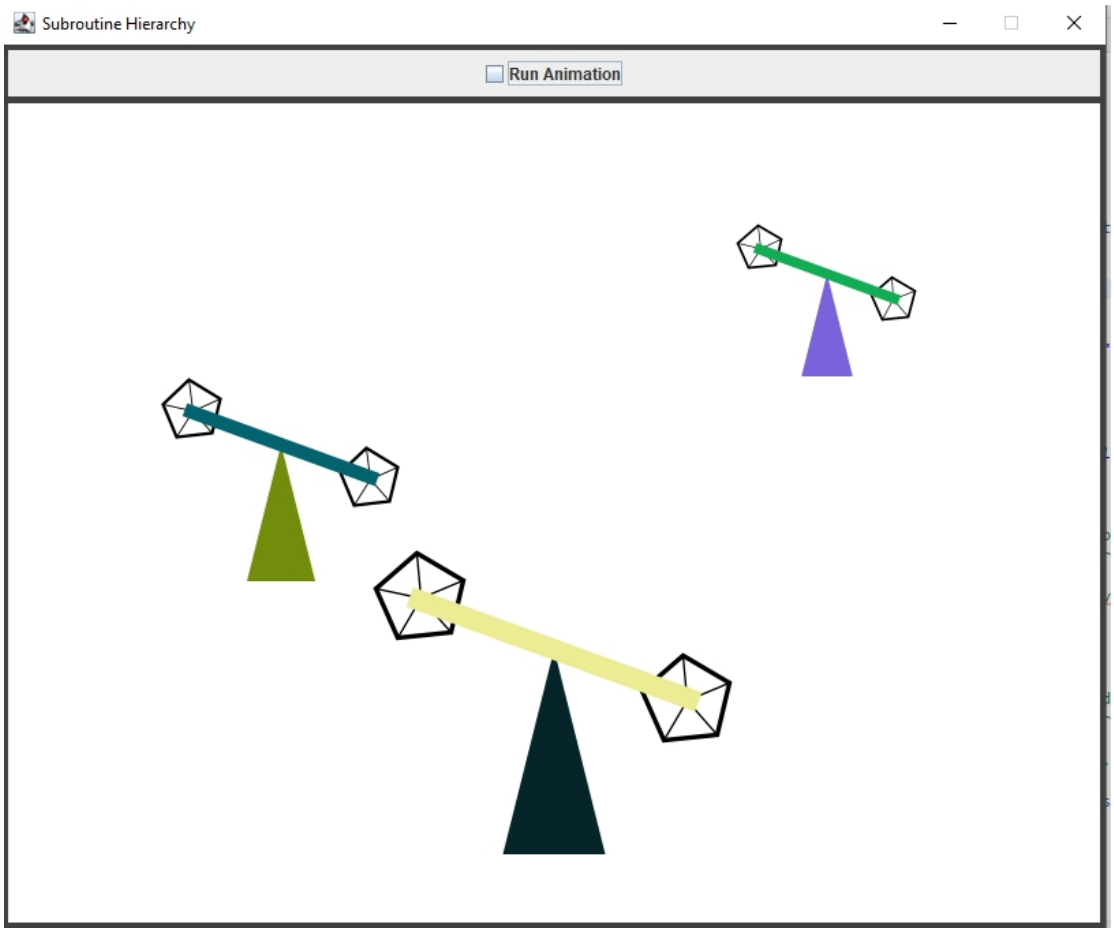
```
 *  Defines a subclass, CompoundObject, of SceneGraphNode to represent
 *  an object that is made up of sub-objects.  Initially, there are no
 *  sub-objects.  Objects are added with the add() method.
 */
private static class CompoundObject extends SceneGraphNode {
   ArrayList<SceneGraphNode> subobjects = 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.
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);
   }
};
private static SceneGraphNode filledPolygon = new SceneGraphNode(){
   void doDraw(Graphics2D g) {
      int R = 50;
      int n = 5;
      int[] xPoints = new int[n];
      int[] yPoints = new int[n];
      for (int i = 0; i < n; i++) {
         xPoints[i] = (int) (R * Math.cos((2 * Math.PI * i) / n));
         yPoints[i] = (int) (R * Math.sin((2 * Math.PI * i) / n));
      }
      Polygon polygon = new Polygon(xPoints, yPoints, n);
      g.translate(0,0);
      g.scale(0.003, 0.003); //skalowanie
      g.setStroke(new BasicStroke(5));
      //g.fillPolygon(polygon); //wypelnienie
      g.drawPolygon(polygon);  //narysowanie
      g.setStroke(new BasicStroke(2));
      for (int i = 0; i < n; i++) {
         g.drawLine(0,0,xPoints[i],yPoints[i]);
      }
   }
};

//---------------------------------- 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 Main() {
   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()) {
```

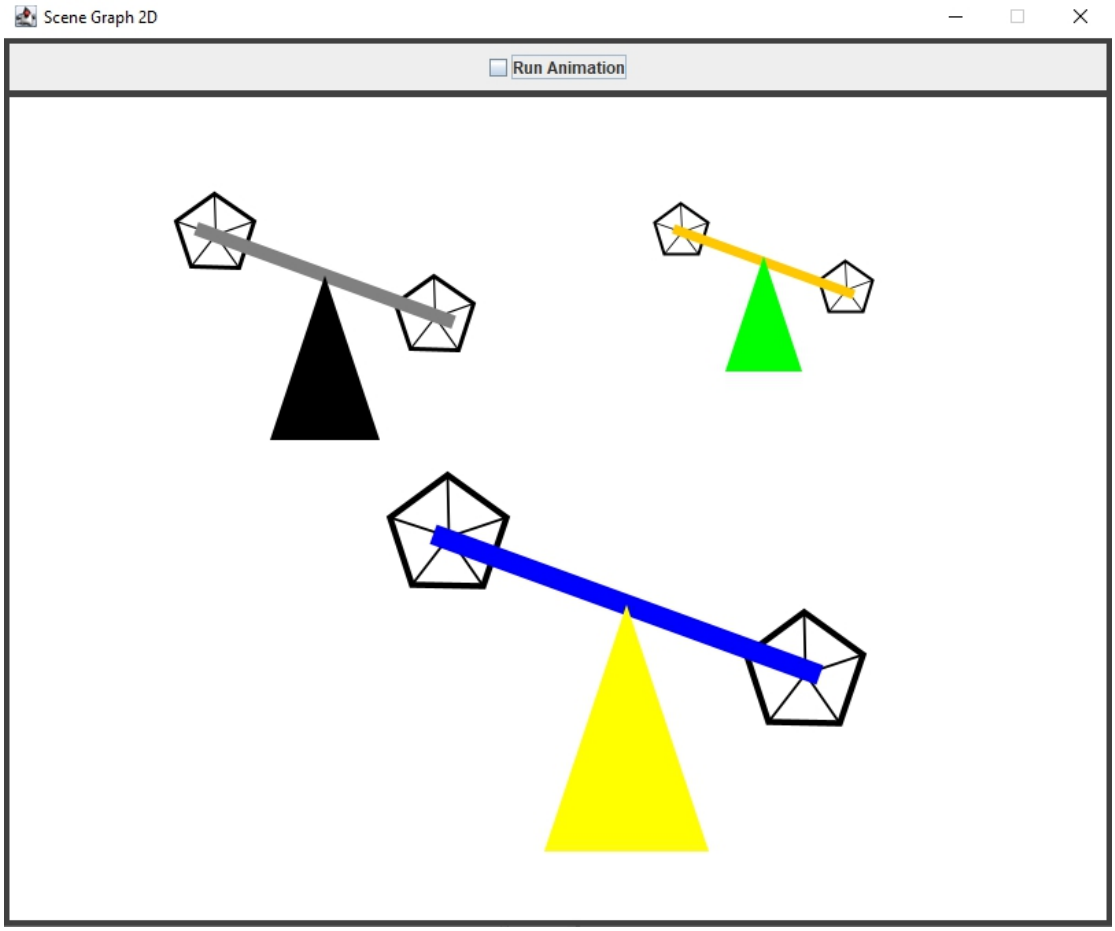
```
            if ( ! timer.isRunning() )
               timer.start();
         }
         else {
            if ( timer.isRunning() )
               timer.stop();
         }
      }
    });
    JPanel top = new JPanel();
    top.add(animationCheck);
    setLayout(new BorderLayout(5,5));
    setBackground(Color.DARK_GRAY);
    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 );
  }
}
```

4. Wyniki działania:

A)



B)



5. Wnioski

Na podstawie otrzymanych wyników można stwierdzić, że:

a) Biblioteka udostępnia nam metody pozwalające wykonanie podstawowych operacji na obiektach
b) Korzystając z języka Java możemy w łatwy sposób narysować proste oraz bardziej skomplikowane wielokąty