SPRAWOZDANIE

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

Laboratorium 3
08.03.2022
Temat:" Modelowanie hierarchiczne w grafice 2D"
Wariant 14kąt

Bartosz Medoń
Informatyka I stopień,
stacjonarne,
4 semestr,
Gr.1a

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,

2. Wykorzystane komendy:

```
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.

    private TransformedObject t1,r1,w11,w12;

    private TransformedObject t2,r2,w21,w22;

    private TransformedObject t3, r3,w31,w32;


    private void createWorld() {

        world = new CompoundObject();

        t1 = new TransformedObject(filledTriangle);
        t1.setScale(0.90, 2).setColor(Color.blue);
        t1.setTranslation(0.5, -2.9);

        r1 = new TransformedObject(filledRect);
        r1.setScale(4, 0.28).setColor(Color.red);
        r1.setTranslation(0.5, -1.0);
        r1.setRotation(168);

        w11 = new TransformedObject(filledPolygon);
        w11.setColor(Color.green);
        w11.setTranslation(-1.4, -0.6);

        w12 = new TransformedObject(filledPolygon);
        w12.setColor(Color.green);
        w12.setTranslation(2.35, -1.4);

        double offsetX = 2.8;
        double offsetY = 2.3;
        double offsetScale = 0.6;
```

```
        t2 = new TransformedObject(filledTriangle);
        t2.setScale(0.6*offsetScale, 1.6*offsetScale).setColor(new
Color(42,120,18));
        t2.setTranslation(-1*offsetScale + offsetX, -2*offsetScale + offsetY);

        r2 = new TransformedObject(filledRect);
        r2.setScale(3.5*offsetScale, 0.25*offsetScale).setColor(Color.red);
        r2.setTranslation(-1*offsetScale + offsetX, -0.5*offsetScale + offsetY);
        r2.setRotation(168);

        w21 = new TransformedObject(filledPolygon);
        w21.setScale(1*offsetScale, 1*offsetScale).setColor(Color.green);
        w21.setTranslation(-2.7*offsetScale + offsetX, -0.15*offsetScale +
offsetY);

        w22 = new TransformedObject(filledPolygon);
        w22.setScale(1*offsetScale, 1*offsetScale).setColor(Color.green);
        w22.setTranslation(0.65*offsetScale + offsetX, -0.85*offsetScale +
offsetY);



        offsetX = -1.4;
        offsetY = 2.6;
        offsetScale = 0.4;

        t3 = new TransformedObject(filledTriangle);
        t3.setScale(1.20*offsetScale, 3.3*offsetScale).setColor(new
Color(145,17,133));
        t3.setTranslation(-1*offsetScale + offsetX, -5.3*offsetScale + offsetY);

        r3 = new TransformedObject(filledRect);
        r3.setScale(6*offsetScale, 0.4*offsetScale).setColor(Color.red);
        r3.setTranslation(-1*offsetScale + offsetX, -2.1*offsetScale + offsetY);
        r3.setRotation(168);

        w31 = new TransformedObject(filledPolygon);
        w31.setScale(1.7*offsetScale, 1.7*offsetScale).setColor(Color.green);
        w31.setTranslation(-3.9*offsetScale + offsetX, -1.5*offsetScale + offsetY);

        w32 = new TransformedObject(filledPolygon);
        w32.setScale(1.7*offsetScale, 1.7*offsetScale).setColor(Color.green);
        w32.setTranslation(1.8*offsetScale + offsetX, -2.7*offsetScale + offsetY);
```

```
        world.add(w11);
        world.add(w12);
        world.add(r1);
        world.add(t1);

        world.add(w21);
        world.add(w22);
        world.add(r2);
        world.add(t2);

        world.add(w31);
        world.add(w32);
        world.add(r3);
        world.add(t3);


    }


    /**
     * 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.

        w12.setRotation(frameNumber*0.75);
        w11.setRotation(frameNumber*0.75);


        w22.setRotation(frameNumber*0.75);
        w21.setRotation(frameNumber*0.75);

        w32.setRotation(frameNumber*0.75);
        w31.setRotation(frameNumber*0.75);
        //rotatingRect.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 g2) {
         int n=14;
         double r = 150,     t=0, k=(Math.PI*2)/n;
         int[] x1 = new int[n];
         int[] y1 = new int[n];
         for (int i=0;i<n;i++)
         {
            x1[i]= (int) (r*Math.sin(t));
            y1[i]= (int) (r*Math.cos(t));
            t+=k;
         }
         Polygon polygon = new Polygon(x1,y1,n);
         g2.scale( 0.0045, 0.0045 );
         g2.fill(polygon);



      }
   };



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

}



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 drawn
 * using subroutines to implement hierarchical modeling.  There is a
 * checkbox that turns the animation on and off.
 */
public class SubroutineHierarchy extends JPanel {

   public static void main(String[] args) {
      JFrame window = new JFrame("Subroutine Hierarchy");
      window.setContentPane( new SubroutineHierarchy() );
      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.
     */
    private void drawWorld(Graphics2D g2) {

        // TODO: Draw the content of the scene.
        F1(g2);

    } // end drawWorld()

    private void updateFrame() {
        frameNumber++;
        // TODO: If other updates are needed for the next frame, do them here.
    }

    private void F1(Graphics2D g2) {
        AffineTransform saveTransform = g2.getTransform();
        Color saveColor = g2.getColor();

        g2.setTransform(saveTransform);
        g2.translate(1, 1);
        int n=15;
        double r = 150,
```

```
        t=0,
        k=(Math.PI*2)/n;
int[] x1 = new int[n];
int[] y1 = new int[n];
for (int i=0;i<n;i++)
{
   x1[i]= (int) (r*Math.sin(t));
   y1[i]= (int) (r*Math.cos(t));
   t+=k;
}


Polygon polygon = new Polygon(x1,y1,n);
g2.translate(1.9, -2.05);
g2.setColor( Color.gray );
g2.rotate( Math.toRadians( frameNumber*0.75 ));
g2.scale( 0.005, 0.005 );
g2.fill(polygon);

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



g2.translate(-0.4, -0);
g2.setColor( Color.gray );
g2.rotate( Math.toRadians( frameNumber*0.75 ));
g2.scale( 0.005, 0.005 );
g2.fill(polygon);

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



g2.translate(0.7,-2.3);
g2.setColor(Color.red);

g2.setStroke(new BasicStroke((float) 0.2));

g2.draw( new Line2D.Double( -1,2.3, 2.2,1.3) );

g2.setTransform(saveTransform);
```

```
g2.translate(0.7, -0.8);
g2.setColor(Color.blue);
triangle1(g2);

g2.setTransform(saveTransform);


g2.setTransform(saveTransform);

g2.scale(0.7, 0.7);
g2.translate(-4.4, 3.2);
g2.setColor( Color.gray );
g2.rotate( Math.toRadians( frameNumber*0.75 ));
g2.scale( 0.005, 0.005 );
g2.fill(polygon);

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


g2.scale(0.7, 0.7);
g2.translate(-1.4, 2.3);
g2.setColor( Color.gray );
g2.rotate( Math.toRadians( frameNumber*0.75 ));
g2.scale( 0.005, 0.005 );
g2.fill(polygon);

g2.setColor(saveColor);


g2.setTransform(saveTransform);

g2.scale(0.7, 1);
g2.translate(-3.5,0.2);
g2.setColor(Color.red);

g2.setStroke(new BasicStroke((float) 0.2));

g2.draw( new Line2D.Double( -0.8,2, 2,1.4) );

g2.setTransform(saveTransform);
```

```
g2.translate(-2.5, 1.7);
g2.setColor(new Color(145,17,133));
g2.scale(0.7, 0.8);
triangle1(g2);


g2.setTransform(saveTransform);


g2.scale(0.6, 0.60);
g2.translate(4.6, 3);
g2.setColor( Color.gray );
g2.rotate( Math.toRadians( frameNumber*0.75 ));
g2.scale( 0.005, 0.005 );
g2.fill(polygon);

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


g2.scale(0.6, 0.6);
g2.translate(1.5, 3.9);
g2.setColor( Color.gray );
g2.rotate( Math.toRadians( frameNumber*0.75 ));
g2.scale( 0.005, 0.005 );
g2.fill(polygon);

g2.setColor(saveColor);


g2.setTransform(saveTransform);

g2.scale(0.6, 0.8);
g2.translate(2.5,0.9);
g2.setColor(Color.blue);

g2.setStroke(new BasicStroke((float) 0.2));

g2.draw( new Line2D.Double( -0.8,2, 2,1.4) );
```

```
        g2.setTransform(saveTransform);

        g2.translate(1.6, 1.9);
        g2.setColor(new Color(42,120,18));
        g2.scale(0.5, 0.6);
        triangle1(g2);

    }

    private void triangle1(Graphics2D g2) {

        g2.translate(0.5, -2);


        Path2D path = new Path2D.Double();
        path.moveTo(-0.5,0);
        path.lineTo(0.5,0);
        path.lineTo(0,2.3);
        path.closePath();
        g2.fill(path);
    }

    private void line1(Graphics2D g2) {
        g2.setColor(Color.red);

        g2.setStroke(new BasicStroke((float) 0.3));

        g2.draw( new Line2D.Double( -1,2.3, 2.2,1.3) );
    }



    //------------------- 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 filledPolygon(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);
   }



   //-------------------------------- 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.
    */
```

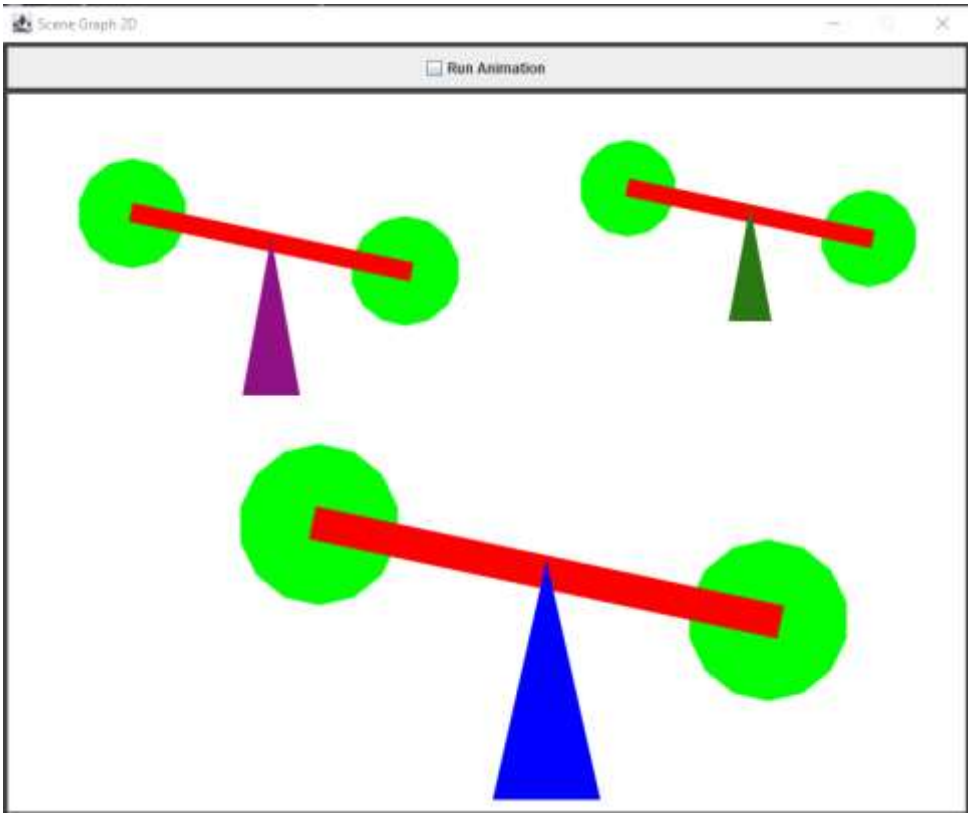
```
   public SubroutineHierarchy() {
      display = new JPanel() {
         protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            Graphics2D g2 = (Graphics2D)g.create();
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
            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);
   }
```

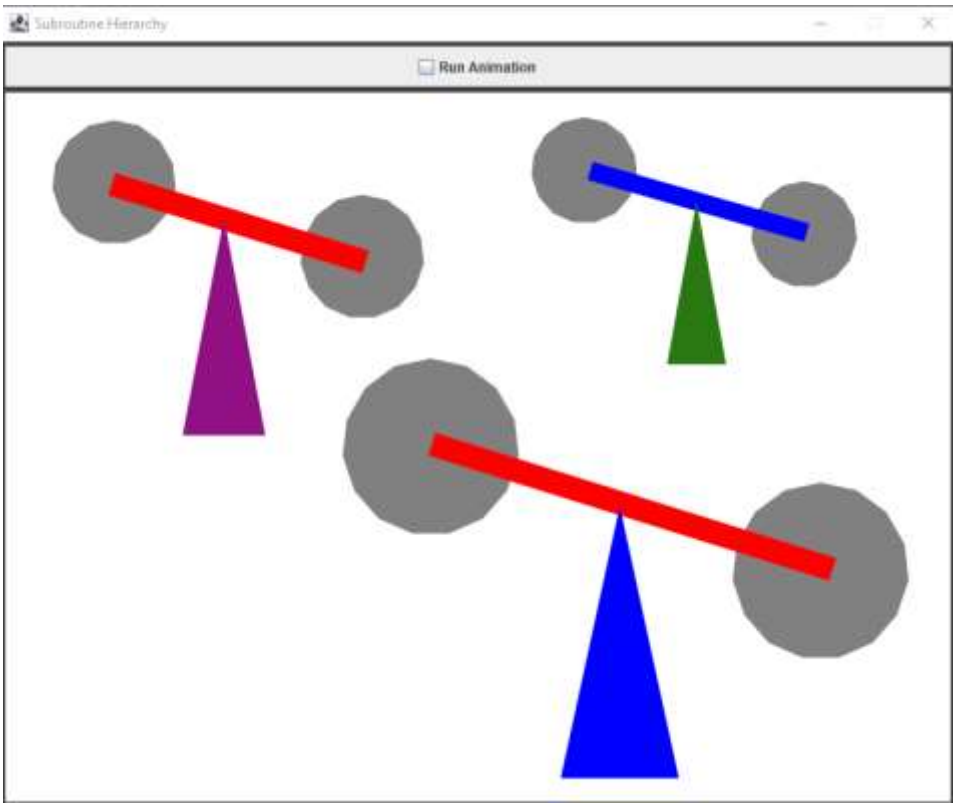
```
   /**
    * Applies a coordinate transform to a Graphics2D graphics context.  The upper left corner of
    * the viewport where the graphics context draws is assumed to be (0,0).  The coordinate
    * transform will make a requested rectangle visible in the drawing area.  The requested
    * limits 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 );
   }

}
```

3. Wynik:





4. Wnioski: Java2D umożliwia tworzenie grafiki, a także animacji