SPRAWOZDANIE

Zajęcia: Grafika komputerowa
Prowadzący: mgr inż. Mikołaj Grygiel

Laboratorium 3
12.03.2025
Temat: "Modelowanie hierarchiczne w grafice 2D"
Wariant 6

Bartłomiej Mędrzak
s61324
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,
na dwa sposoby:
(a) używając hierarchię funkcje (sposób subroutinowy)
(b) tworząc graf sceny (sposób obiektowy).

2. Wprowadzane dane:

Aby odtworzyć obrazek podany w zadaniu należało dodać różne figury geometryczne. Należało również zadbać o poprawną figurę „koła” zgodną z wariantem.

3. Wykorzystane komendy:

a) **Hierarchia**

```
private void Line(Graphics2D g2, double translate_x, double translate_y) {
    g2.setColor(Color.RED);
    g2.translate(translate_x, translate_y);
    g2.rotate(-Math.PI / 8);
    g2.scale(sx:2.29, sy:0.14);
    filledRect(g2);
}

private void R_Line(Graphics2D g2, double skala_x, double skala_y, double translate_x, double translate_y,
                    Color color) {
    AffineTransform saveTransform = g2.getTransform();
    g2.scale(skala_x, skala_y);
    Line(g2, translate_x, translate_y);
    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 }, nPoints:3);
    g2.setTransform(saveTransform);
}
```

```
private void drawWorld(Graphics2D g2) {
    rotatingPolygon(g2, r:100, -1.02, -0.05); // Graphics2D, promień, translate_x, translate_y
    rotatingPolygon(g2, r:100, translate_x:1.04, -0.9);
    rotatingPolygon(g2, r:80, -1.3, translate_y:1.41);
    rotatingPolygon(g2, r:80, -3.12, translate_y:2.22);
    rotatingPolygon(g2, r:60, translate_x:0.85, translate_y:2.04);
    rotatingPolygon(g2, r:60, translate_x:2.12, translate_y:1.44);

    R_Line(g2, skala_x:1, skala_y:1.05, translate_x:0, -0.46, Color.RED); // Graphics2D, skala_x, skala_y, translate_x, translate_y, color
    R_Line(g2, skala_x:0.85, skala_y:0.95, -2.6, translate_y:1.87, Color.RED);
    R_Line(g2, skala_x:0.6, skala_y:0.7, translate_x:2.47, translate_y:2.47, Color.RED);

    Triangle(g2, skala_x:0.4, skala_y:0.5, translate_x:0, -2, Color.BLUE); // Graphics2D, skala_x, skala_y, translate_x, translate_y, color
    Triangle(g2, skala_x:0.25, skala_y:0.35, -2.25, translate_y:0.75, new Color(r:200, g:21, b:132));
    Triangle(g2, skala_x:0.15, skala_y:0.25, translate_x:1.5, translate_y:1, new Color(r:0, g…128, 0));
}// end drawWorld()
```

```
private void rotatingPolygon(Graphics2D g2, double r, double translate_x, double translate_y) {

    AffineTransform saveTransform = g2.getTransform();
    Color saveColor = g2.getColor();
    g2.setTransform(saveTransform);
    g2.setStroke(new BasicStroke(width:2));

    int n = 10;
    double t = 0, ang = (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 += ang;
    }

    Polygon polygon = new Polygon(x1, y1, n);
    g2.translate(translate_x, translate_y);
    g2.setColor(Color.black);
    g2.rotate(-Math.toRadians(frameNumber * 2));
    g2.scale(sx:0.00475, sy:0.00475);

    for (int i = 0; i < n; i++) {
        g2.drawLine(x1[i], y1[i], x2:0, y2:0);
    }
    g2.draw(polygon);
    g2.setColor(saveColor);
    g2.setTransform(saveTransform);
}
```

b) **Graf sceny**

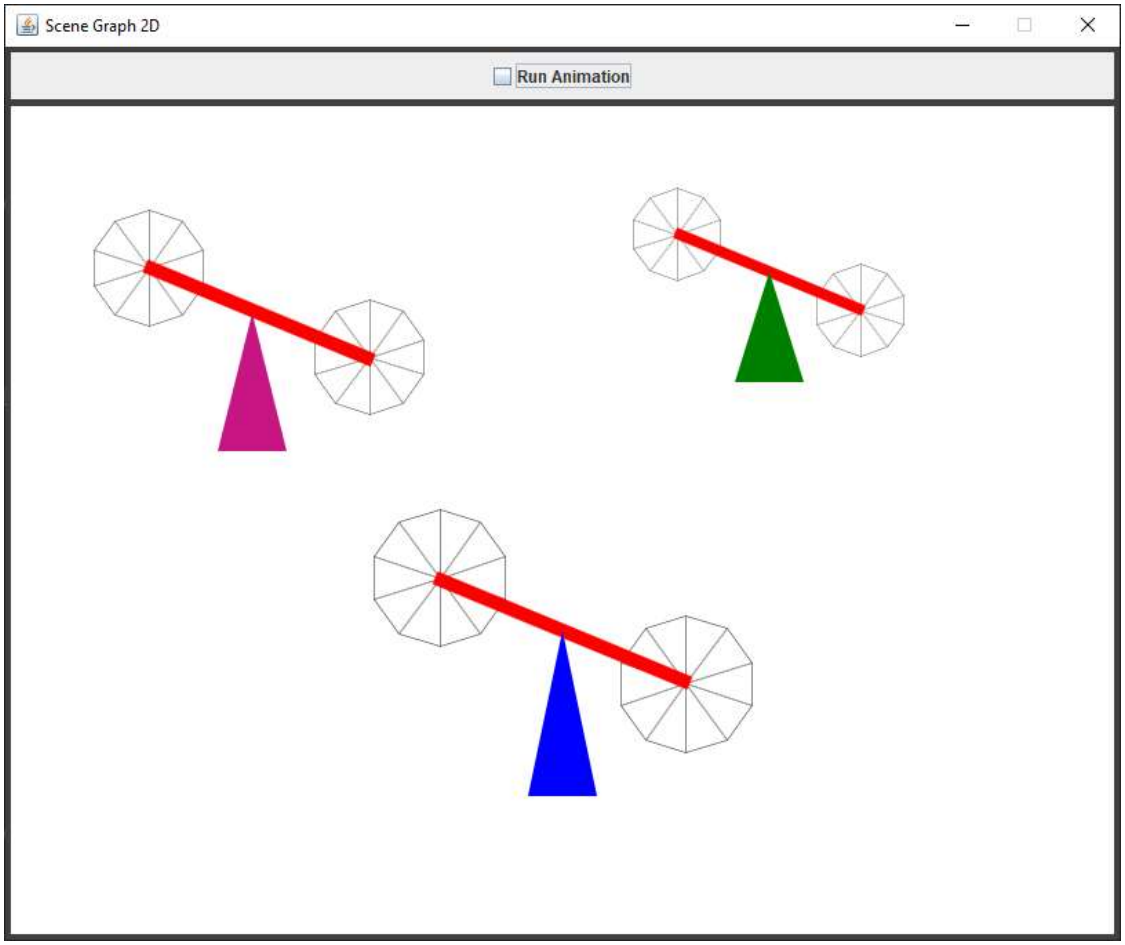
```
private static SceneGraphNode filledPolygon = new SceneGraphNode() {
    void doDraw(Graphics2D g) {

        int n = 10;
        double t = 0, ang = (Math.PI * 2) / n;

        int[] x1 = new int[n];
        int[] y1 = new int[n];

        for (int i = 0; i < n; i++) {
            x1[i] = (int) (350 * Math.sin(t));
            y1[i] = (int) (350 * Math.cos(t));
            t += ang;
        }

        Polygon polygon = new Polygon(x1, y1, n);
        g.setStroke(new BasicStroke(width:4));
        g.scale(sx:0.00475, sy:0.00475);

        for (int i = 0; i < n; i++) {
            g.drawLine(x1[i], y1[i], x2:0, y2:0);
        }

        g.draw(polygon);

    }
};
```

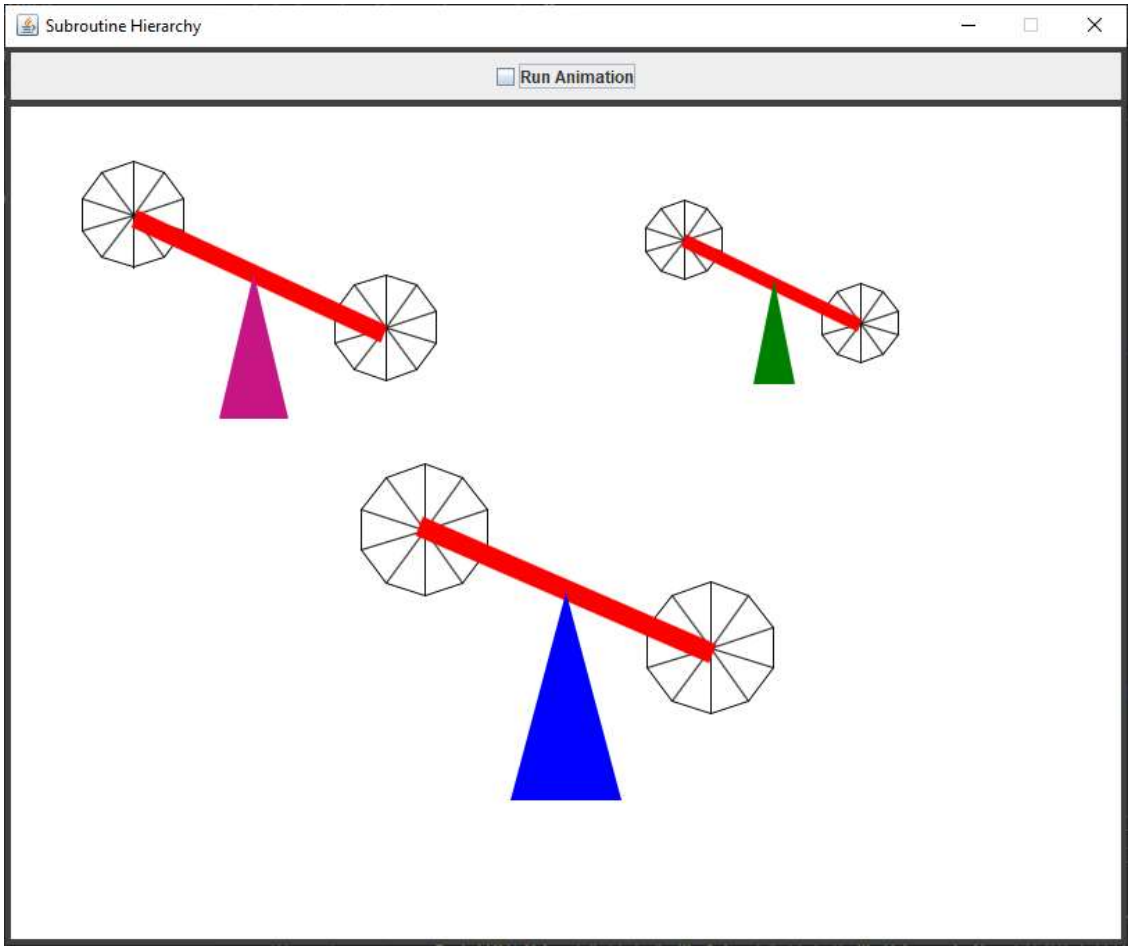
https://github.com/castehard33/Grafika_Komputerowa/tree/main/3%20Modelowanie%20hierarchiczne%20w%20grafice%202D

```
private void createWorld() {

    world = new CompoundObject();

    P1 = new TransformedObject(filledPolygon);
    P2 = new TransformedObject(filledPolygon);
    P3 = new TransformedObject(filledPolygon);
    P4 = new TransformedObject(filledPolygon);
    P5 = new TransformedObject(filledPolygon);
    P6 = new TransformedObject(filledPolygon);
    L1 = new TransformedObject(filledRect);
    L2 = new TransformedObject(filledRect);
    L3 = new TransformedObject(filledRect);
    T1 = new TransformedObject(filledTriangle);
    T2 = new TransformedObject(filledTriangle);
    T3 = new TransformedObject(filledTriangle);

    P1.setScale(sx:0.3, sy:0.3).setTranslation(-0.889, -0.42);
    P2.setScale(sx:0.3, sy:0.3).setTranslation(dx:0.899, -1.189);
    P3.setScale(sx:0.25, sy:0.25).setTranslation(-3, dy:1.825);
    P4.setScale(sx:0.25, sy:0.25).setTranslation(-1.4, dy:1.18);
    P5.setScale(sx:0.2, sy:0.2).setTranslation(dx:0.83, dy:2.07);
    P6.setScale(sx:0.2, sy:0.2).setTranslation(dx:2.16, dy:1.52);
    T1.setScale(sx:0.5, sy:1.2).setTranslation(dx:0, -2).setColor(Color.BLUE);
    L1.setRotation(-22.5).setScale(sx:2, sy:0.1).setTranslation(dx:0, -0.8).setColor(Color.RED);
    L2.setRotation(-22.5).setScale(sx:1.79, sy:0.1).setTranslation(-2.2, dy:1.50).setColor(Color.RED);
    L3.setRotation(-22.5).setScale(sx:1.48, sy:0.08).setTranslation(dx:1.5, dy:1.8).setColor(Color.RED);
    T2.setScale(sx:0.5, sy:1).setTranslation(-2.25, dy:0.5).setColor(new Color(r:200, g:21, b:132));
    T3.setScale(sx:0.5, sy:0.8).setTranslation(dx:1.5, dy:1).setColor(new Color(r:0, g:128, b:0));

    world.add(P1);
    world.add(P2);
    world.add(P3);
    world.add(P4);
    world.add(P5);
    world.add(P6);
    world.add(L1);
    world.add(L2);
    world.add(L3);
    world.add(T1);
    world.add(T2);
    world.add(T3);
```

```
// TODO: Define global variables to represent animated objects in the scene.
private TransformedObject T1, T2, T3;
private TransformedObject L1, L2, L3;
private TransformedObject P1, P2, P3, P4, P5, P6;
```

4. Wynik działania:


Scene Graph 2D
Run Animation


Subroutine Hierarchy
Run Animation

5. Wnioski:

Na podstawie otrzymanego wyniku można stwierdzić, że obie zaimplementowane metody – subroutinowa oraz obiektowa z wykorzystaniem grafu sceny – pozwalają na skuteczne modelowanie hierarchiczne i animację obiektów 2D. Podejście subroutinowe, choć początkowo może wydawać się prostsze w implementacji dla mniejszych scen, wymaga starannego zarządzania transformacjami i stanem graficznym, co może prowadzić do błędów przy większej złożoności.

Natomiast metoda oparta na grafie sceny, dzięki swojej obiektowej naturze, oferuje znacznie lepszą organizację kodu, hermetyzację stanu poszczególnych elementów (np. transformacji, koloru) oraz większą modularność i łatwość rozbudowy sceny. Implementacja animacji, w szczególności obrotu wielokątów, okazała się bardziej intuicyjna w podejściu obiektowym, gdzie każdy animowany obiekt mógł zarządzać własnym stanem rotacji, aktualizowanym w dedykowanej metodzie updateFrame. Graf sceny ułatwia również ponowne wykorzystanie zdefiniowanych kształtów i złożonych obiektów, co jest kluczowe przy tworzeniu bardziej skomplikowanych kompozycji, jak w przypadku wielokrotnych instancji huśtawek z obracającymi się kołami.