**SPRAWOZDANIE**

Zajęcia: Grafika komputerowa

Prowadzący: prof. dr hab. Vasyl Martsenyuk

**Laboratorium 3**

**Data 08.03.2022**

**Temat: Modelowanie hierarchiczne w grafice 2D**

**Wariant 15**

Natalia Pierzchała

Informatyka I stopień,

stacjonarne,

4 semestr,

Gr. 2a

# 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:

1. używając hierarchię funkcje (sposób subroutinowy)
2. tworząc graf sceny (sposób obiektowy). W tym celu proponuję do pobrania odpowiedni pliki

# Wprowadzane dane i wykorzystane komendy:

1. Sposób subroutinowy

**private** **void** S\_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** 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(2.3, 0.15);

*filledRect*(g2);

}

**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** r, **double** translate\_x ,**double** translate\_y) // polygon

{

AffineTransform saveTransform = g2.getTransform();

Color saveColor = g2.getColor();

g2.setTransform(saveTransform);

g2.setStroke(**new** BasicStroke(2));

**int** n=15;

**double** 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(translate\_x,translate\_y);

g2.setColor( Color.***black*** );

g2.rotate( Math.*toRadians*( frameNumber\*0.75 ));

g2.scale( 0.005, 0.005 );

**for**(**int** i=0;i<n;i++)

{

g2.drawLine( x1[i],y1[i],0,0 );

}

g2.draw(polygon);

g2.setColor(saveColor);

g2.setTransform(saveTransform);

}

1. Graf 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;

**private** **void** createWorld() {

world = **new** CompoundObject();

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);

}

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);

**private** **static** SceneGraphNode *F\_Polygon* = **new** SceneGraphNode()

{**void** doDraw(Graphics2D g){

**int** n=15;

**double** 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**) (350\*Math.*sin*(t));

y1[i]= (**int**) (350\*Math.*cos*(t));

t+=k;

}

Polygon polygon = **new** Polygon(x1,y1,n);

g.setStroke(**new** BasicStroke(4));

g.scale( 0.005, 0.005 );

**for**(**int** i=0;i<n;i++)

{

g.drawLine( x1[i],y1[i],0,0 );

}

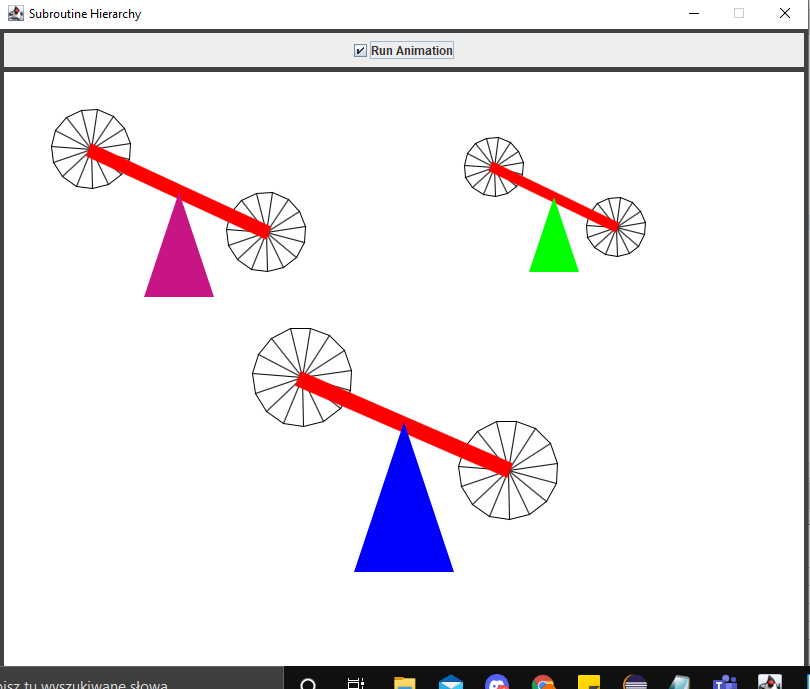
g.draw(polygon);

}

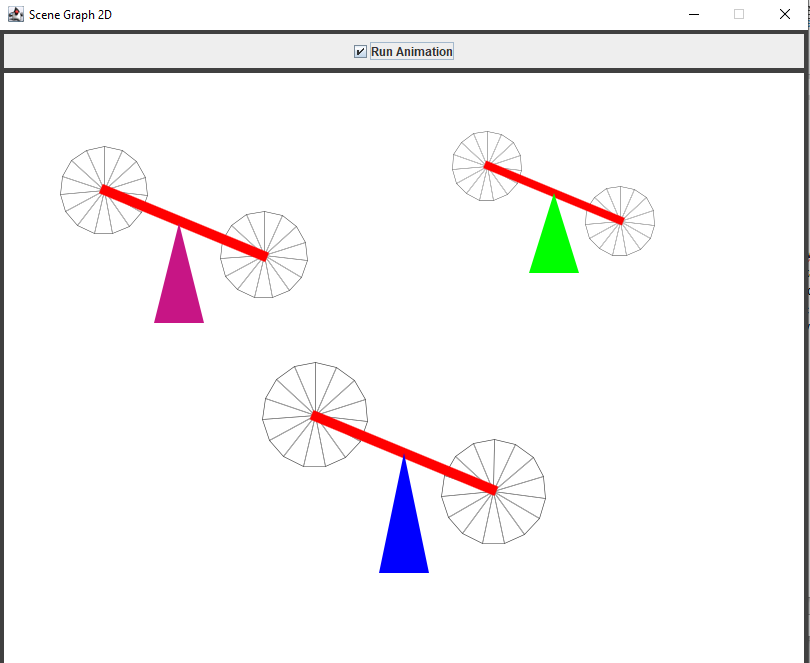
};

# Wyniki działania

1. Sposób subroutinowy



1. Graf sceny



# Wnioski

Można stwierdzić, że dzięki podejściu subroutinowym i podejściu obiektowym można stworzyć scenę hierarchiczną, co pozwala na uzyskanie hierarchii w figurach na scenie. Taką scenę można tworzyć w języku Java oraz JavaScript.