МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 3

з дисципліни ”Математичні та алгоритмічні основи комп’ютерної графіки”

Виконав(ла)
студент(ка) III курсу
групи КП - 81
Івахненко Маргарита Василівна
(*прізвище, ім’я, по батькові*)

Київ 2021

Постановка завдання

За допомогою примітивів JavaFX максимально реально зобразити персонажа за
варіантом та виконати його 2D анімацію. Для анімації скористатися стандартними
засобами бібліотеки JavaFX.

Обов’язковою є реалізація таких видів анімації:

1) переміщення;

2) поворот;

3) масштабування.

Студентам пропонується скористатися розглянутими класами для читання, обробки
та збереження зображень формату .bmp з метою використання рисунку для створення
траєкторії руху або меж, в яких дозволений рух об’єктів. В даному випадку
рекомендується використовувати кольори великої контрастності для різних призначень
(наприклад, чорний колір відповідатиме за траєкторію руху, а інші кольори – заборонятимуть рух)

Тексти коду програми

PrintingImage.java

```
package sample;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;

import javafx.animation.ParallelTransition;
import javafx.animation.PathTransition;
import javafx.animation.RotateTransition;
import javafx.animation.ScaleTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
import javafx.stage.Stage;
import javafx.util.Duration;

public class PrintingImage extends Application {

   private HeaderBitmapImage image;
   private int numberOfPixels;

   public PrintingImage() {}

   public PrintingImage(HeaderBitmapImage image) {
       this.image = image;
   }

   @Override
   public void start(Stage primaryStage) throws Exception{

ReadingImageFromFile.loadBitmapImage("source/trajectory.bmp");
       this.image = ReadingImageFromFile.pr.image;
       int width = (int) this.image.getWidth();
       int height = (int) this.image.getHeight();
       int half = (int) image.getHalfOfWidth();

       Group root = new Group();
       Scene scene = new Scene(root, width, height + 200);
       scene.setFill(Color.BLACK);

       int let = 0;
       int let1 = 0;
       int let2 = 0;
       char[][] map = new char[width][height];
```

```
       BufferedInputStream reader = new BufferedInputStream (new
FileInputStream("pixels.txt"));

       for(int i = 0; i < height; i++) {
           for(int j = 0; j < half; j++) {
               let = reader.read();
               let1 = let;
               let2 = let;
               let1 = let1&(0xf0);
               let1 = let1 >> 4;
               let2 = let2&(0x0f);

               if(j * 2 < width) {
                   if (returnPixelColor(let1) == "BLACK")
                   {
                       map[j * 2][height - 1 - i] = '1';
                       numberOfPixels++;
                   }
                   else
                   {
                       map[j * 2][height - 1 - i] = '0';
                   }
               }
               if(j * 2 + 1 < width)
               {
                   if (returnPixelColor(let2) == "BLACK")
                   {
                       map[j * 2 + 1][height - 1 - i] = '1';
                       numberOfPixels++;
                   }
                   else
                   {
                       map[j * 2 + 1][height - 1 - i] = '0';
                   }
               }
           }
       }

       primaryStage.setScene(scene);
       primaryStage.show();

       reader.close();

       int[][] black;
       black = new int[numberOfPixels][2];
       int lich = 0;

       BufferedOutputStream writer = new BufferedOutputStream (new
FileOutputStream("map.txt"));
       for(int i = 0; i < height; i++) {
           for(int j = 0; j < width; j++) {
```

```
                if (map[j][i] == '1') {
                    black[lich][0] = j;
                    black[lich][1] = i;
                    lich++;
                }
                writer.write(map[j][i]);
            }
            writer.write(10);
        }
        writer.close();

        System.out.println("number of black color pixels = " +
numberOfPixels);

        Group g = getFigure();
        root.getChildren().add(g);

        Path path2 = new Path();
        for (int l=0; l<numberOfPixels-1; l++)
        {
            path2.getElements().addAll(
                    new MoveTo(black[l][0],black[l][1]),
                    new LineTo (black[l+1][0],black[l+1][1])
            );
        }

        PathTransition pathTransition = new PathTransition();
        pathTransition.setDuration(Duration.millis(5000));
        pathTransition.setPath(path2);
        pathTransition.setNode(g);

        RotateTransition rotateTransition = new
RotateTransition(Duration.millis(500), g);
        rotateTransition.setByAngle(360);
        rotateTransition.setCycleCount(10);
        rotateTransition.setAutoReverse(true);

        ScaleTransition scaleTransition =
                new ScaleTransition(Duration.seconds(4.0), g);
        scaleTransition.setToX(-2f);
        scaleTransition.setToY(-2f);


        ParallelTransition parallelTransition = new
ParallelTransition();
        parallelTransition.getChildren().addAll(
                pathTransition,
                rotateTransition,
                scaleTransition
        );
```

```
        parallelTransition.play();
    }

    private String returnPixelColor (int color) {
        String col = "BLACK";
        switch(color) {
            case 0: return "BLACK";
            case 1: return "LIGHTCORAL";
            case 2: return "GREEN";
            case 3: return "BROWN";
            case 4: return "BLUE";
            case 5: return "MAGENTA";
            case 6: return "CYAN";
            case 7: return "LIGHTGRAY";
            case 8: return "DARKGRAY";
            case 9: return "RED";
            case 10:return "LIGHTGREEN";
            case 11:return "YELLOW";
            case 12:return "LIGHTBLUE";
            case 13:return "LIGHTPINK";
            case 14:return "LIGHTCYAN";
            case 15:return "WHITE";
        }
        return col;
    }

    public static void main(String[] args) {
        launch(args);
    }

    private Group getFigure() {
        Group g = new Group();

        final Circle cir = new Circle ();
        cir.setCenterX(64);
        cir.setCenterY(64);
        cir.setRadius(64);
        cir.setFill(Color.YELLOW);
        g.getChildren().add(cir);

        final Group smile = new Group();

        Arc smileMouth = new Arc();
        smileMouth.setCenterX(64.0f);
        smileMouth.setCenterY(84.0f);
        smileMouth.setRadiusX(45.0f);
        smileMouth.setRadiusY(25.0f);
        smileMouth.setStartAngle(180.0f);
        smileMouth.setLength(180.0f);
        smileMouth.setType(ArcType.ROUND);
        smileMouth.setFill(Color.BLACK);
```

```
        smile.getChildren().add(smileMouth);

        Arc smileTeeth = new Arc();
        smileTeeth.setCenterX(64.0f);
        smileTeeth.setCenterY(84.0f);
        smileTeeth.setRadiusX(45.0f);
        smileTeeth.setRadiusY(15.0f);
        smileTeeth.setStartAngle(180.0f);
        smileTeeth.setLength(180.0f);
        smileTeeth.setType(ArcType.ROUND);
        smileTeeth.setFill(Color.WHITE);
        smile.getChildren().add(smileTeeth);

        g.getChildren().add(smile);


        Group leftEye = new Group();

        Arc leftEye1 = new Arc();
        leftEye1.setCenterX(40.0f);
        leftEye1.setCenterY(55.0f);
        leftEye1.setRadiusX(17.0f);
        leftEye1.setRadiusY(27.0f);
        leftEye1.setStartAngle(0.0f);
        leftEye1.setLength(200.0f);
        leftEye1.setType(ArcType.ROUND);
        leftEye1.setFill(Color.WHITE);
        leftEye.getChildren().add(leftEye1);

        Arc leftEye2 = new Arc();
        leftEye2.setCenterX(40.0f);
        leftEye2.setCenterY(55.0f);
        leftEye2.setRadiusX(12.0f);
        leftEye2.setRadiusY(18.0f);
        leftEye2.setStartAngle(0.0f);
        leftEye2.setLength(200.0f);
        leftEye2.setType(ArcType.CHORD);
        leftEye2.setFill(Color.BLUE);
        leftEye.getChildren().add(leftEye2);

        Arc leftEye3 = new Arc();
        leftEye3.setCenterX(40.0f);
        leftEye3.setCenterY(55.0f);
        leftEye3.setRadiusX(10.0f);
        leftEye3.setRadiusY(12.0f);
        leftEye3.setStartAngle(0.0f);
        leftEye3.setLength(200.0f);
        leftEye3.setType(ArcType.CHORD);
        leftEye3.setFill(Color.BLACK);
        leftEye.getChildren().add(leftEye3);
```

```
Circle leftEye4 = new Circle();
leftEye4.setCenterX(46.0f);
leftEye4.setCenterY(48.0f);
leftEye4.setRadius(2.5f);
leftEye4.setFill(Color.WHITE);
leftEye.getChildren().add(leftEye4);

g.getChildren().add(leftEye);

Group rightEye = new Group();

Arc rightEye1 = new Arc();
rightEye1.setCenterX(95.0f);
rightEye1.setCenterY(55.0f);
rightEye1.setRadiusX(17.0f);
rightEye1.setRadiusY(27.0f);
rightEye1.setStartAngle(-20.0f);
rightEye1.setLength(200.0f);
rightEye1.setType(ArcType.ROUND);
rightEye1.setFill(Color.WHITE);
rightEye.getChildren().add(rightEye1);

Arc rightEye2 = new Arc();
rightEye2.setCenterX(95.0f);
rightEye2.setCenterY(55.0f);
rightEye2.setRadiusX(12.0f);
rightEye2.setRadiusY(18.0f);
rightEye2.setStartAngle(-20.0f);
rightEye2.setLength(200.0f);
rightEye2.setType(ArcType.CHORD);
rightEye2.setFill(Color.BLUE);
rightEye.getChildren().add(rightEye2);

Arc rightEye3 = new Arc();
rightEye3.setCenterX(95.0f);
rightEye3.setCenterY(55.0f);
rightEye3.setRadiusX(10.0f);
rightEye3.setRadiusY(14.0f);
rightEye3.setStartAngle(-20.0f);
rightEye3.setLength(200.0f);
rightEye3.setType(ArcType.CHORD);
rightEye3.setFill(Color.BLACK);
rightEye.getChildren().add(rightEye3);

Circle rightEye4 = new Circle();
rightEye4.setCenterX(100.0f);
rightEye4.setCenterY(48.0f);
rightEye4.setRadius(2.5f);
rightEye4.setFill(Color.WHITE);
rightEye.getChildren().add(rightEye4);
```

```
        g.getChildren().add(rightEye);

        Arc leftEyeBrow = new Arc();
        leftEyeBrow.setCenterX(40.0f);
        leftEyeBrow.setCenterY(30.0f);
        leftEyeBrow.setRadiusX(20.0f);
        leftEyeBrow.setRadiusY(10.0f);
        leftEyeBrow.setStartAngle(60.0f);
        leftEyeBrow.setLength(100.0f);
        leftEyeBrow.setStrokeWidth(2.5f);
        leftEyeBrow.setType(ArcType.OPEN);
        leftEyeBrow.setStroke(Color.BLACK);
        leftEyeBrow.setFill(Color.TRANSPARENT);

        g.getChildren().add(leftEyeBrow);

        Arc rightEyeBrow = new Arc();
        rightEyeBrow.setCenterX(90.0f);
        rightEyeBrow.setCenterY(30.0f);
        rightEyeBrow.setRadiusX(20.0f);
        rightEyeBrow.setRadiusY(10.0f);
        rightEyeBrow.setStartAngle(25.0f);
        rightEyeBrow.setLength(100.0f);
        rightEyeBrow.setStrokeWidth(2.5f);
        rightEyeBrow.setType(ArcType.OPEN);
        rightEyeBrow.setStroke(Color.BLACK);
        rightEyeBrow.setFill(Color.TRANSPARENT);

        g.getChildren().add(rightEyeBrow);

        return g;
    }
}
```

HeaderBitmapImage.java

```
package sample;

public class HeaderBitmapImage {
   private short type;
   private long size;
   private short reserveField1;
   private short reserveField2;
   private long offset;
   private long sizeOfHeader;
   private long width;
   private long height;
   private short numberOfColorPlanes;
   private short bitsCount;
   private long compression;
   private long sizeOfCompImage;
   private long horizontalResolution;
   private long verticalResolution;
   private long numbOfUsedColors;
   private long numbOfImportantColors;

   private long halfOfWidth;

   public HeaderBitmapImage () {}

   public short getType () {
       return type;
   }

   public void setType (short type)
   {
       this.type = type;
   }

   public long getSize ()
   {
       return size;
   }

   public void setSize (long size)
   {
       this.size = size;
   }

   public short getReserveField1 ()
   {
       return reserveField1;
   }
```

```
    public void setReserveField1 (short reserveField1)
    {
        this.reserveField1 = reserveField1;
    }

    public short getReserveField2 ()
    {
        return reserveField2;
    }

    public void setReserveField2 (short reserveField2)
    {
        this.reserveField2 = reserveField2;
    }

    public long getOffset ()
    {
        return offset;
    }

    public void setOffset (long offset)
    {
        this.offset = offset;
    }

    public long getSizeOfHeader ()
    {
        return sizeOfHeader;
    }

    public void setSizeOfHeader (long sizeOfHeader)
    {
        this.sizeOfHeader = sizeOfHeader;
    }

    public long getWidth ()
    {
        return width;
    }

    public void setWidth (long width)
    {
        this.width = width;
    }

    public long getHeight ()
    {
        return height;
    }

    public void setHeight (long height)
```

```
    {
        this.height = height;
    }

    public short getNumberOfColorPlanes ()
    {
        return numberOfColorPlanes;
    }

    public void setNumberOfColorPlanes (short numberOfColorPlanes)
    {
        this.numberOfColorPlanes = numberOfColorPlanes;
    }


    public short getBitsCount ()
    {
        return bitsCount;
    }

    public void setBitsCount (short bitsCount)
    {
        this.bitsCount = bitsCount;
    }

    public long getCompression ()
    {
        return compression;
    }

    public void setCompression (long compression)
    {
        this.compression = compression;
    }

    public long getSizeOfCompImage ()
    {
        return sizeOfCompImage;
    }

    public void setSizeOfCompImage (long sizeOfCompImage)
    {
        this.sizeOfCompImage = sizeOfCompImage;
    }

    public long getHorizontalResolution ()
    {
        return horizontalResolution;
    }

    public void setHorizontalResolution (long horizontalResolution)
```

```
    {
        this.horizontalResolution = horizontalResolution;
    }

    public long getVerticalResolution ()
    {
        return verticalResolution;
    }

    public void setVerticalResolution (long verticalResolution)
    {
        this.verticalResolution = verticalResolution;
    }

    public long getNumbOfUsedColors ()
    {
        return numbOfUsedColors;
    }

    public void setNumbOfUsedColors (long numbOfUsedColors)
    {
        this.numbOfUsedColors = numbOfUsedColors;
    }

    public long getNumbOfImportantColors ()
    {
        return numbOfImportantColors;
    }

    public void setNumbOfImportantColors (long
numbOfImportantColors)
    {
        this.numbOfImportantColors = numbOfImportantColors;
    }

    public long getHalfOfWidth ()
    {
        return halfOfWidth;
    }

    public void setHalfOfWidth (long halfOfWidth)
    {
        this.halfOfWidth = halfOfWidth;
    }

    public void setValues (short type, long size, short resF1, short
resF2, long offs,
                           long sHeader, long w, long h, short
nColPan, short bCount, long compr, long sComp,
                           long hRes, long vRes, long nUsCol, long
nImpCol, long half )
```

```
   {
        setType(type);
        setSize(size);
        setReserveField1(resF1);
        setReserveField2(resF2);
        setOffset(offs);
        setSizeOfHeader(sHeader);
        setWidth(w);
        setHeight(h);
        setNumberOfColorPlanes(nColPan);
        setBitsCount(bCount);
        setCompression(compr);
        setSizeOfCompImage(sComp);
        setHorizontalResolution(hRes);
        setVerticalResolution(vRes);
        setNumbOfUsedColors(nUsCol);
        setNumbOfImportantColors(nImpCol);
        setHalfOfWidth(half);
   }
}
```

ReadingImageFromFile.java

```
package sample;

import java.io.*;

public class ReadingImageFromFile {
   public static PrintingImage pr;

   public static void loadBitmapImage(String filename) throws
IOException
   {
        int line;
        BufferedInputStream reader = new BufferedInputStream(new
FileInputStream(filename));
        BufferedOutputStream writer = new BufferedOutputStream(new
FileOutputStream("primer_bmp.txt"));

        while ((line = reader.read()) != -1) {
            writer.write(line);
        }

        reader.close();
        writer.close();

        BufferedInputStream reader1 = new BufferedInputStream(new
FileInputStream("primer_bmp.txt"));
```

```
       ReadingHeaderFromBitmapImage reading = new
ReadingHeaderFromBitmapImage();
       HeaderBitmapImage hbi = new HeaderBitmapImage();
       hbi = reading.Reading(reader1);
       pr = reading.pr;

       System.out.println("type = " + hbi.getType());
       System.out.println("size = " + hbi.getSize());
       System.out.println("reserve field 1 = " +
hbi.getReserveField1());
       System.out.println("reserve field 2 = " +
hbi.getReserveField2());
       System.out.println("offset = " + hbi.getOffset());
       System.out.println("size of header = " +
hbi.getSizeOfHeader());
       System.out.println("width = " + hbi.getWidth());
       System.out.println("height = " + hbi.getHeight());
       System.out.println("number of planes = " +
hbi.getNumberOfColorPlanes());
       System.out.println("number of bits = " +
hbi.getBitsCount());
       System.out.println("type of compression = " +
hbi.getCompression());
       System.out.println("size of image after compression = " +
hbi.getSizeOfCompImage());
       System.out.println("horizontal resolution = " +
hbi.getHorizontalResolution());
       System.out.println("vertical resolution = " +
hbi.getVerticalResolution());
       System.out.println("number of used colors = " +
hbi.getNumbOfUsedColors());
       System.out.println("number of important colors = " +
hbi.getNumbOfImportantColors());

       System.out.println("half of width = " +
hbi.getHalfOfWidth());

       reader1.close();
   }

   public static void main(String[] args) throws IOException { }
}
```

ReadingHeaderFromBitmapImage.java

```
package sample;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class ReadingHeaderFromBitmapImage {
   public PrintingImage pr;

   public ReadingHeaderFromBitmapImage() {}

   public HeaderBitmapImage Reading (BufferedInputStream reader1)
throws IOException
   {
       HeaderBitmapImage hbi = new HeaderBitmapImage();

       int line;
       int i = 0;
       short type = 0;
       long size = 0;
       short res1 = 0;
       short res2 = 0;
       long offset = 0;
       long header = 0;
       long width = 0;
       long height = 0;
       short numbPanel = 0;
       short bitCount = 0;
       long compr = 0;
       long sCompIm = 0;
       long hRes = 0;
       long vRes = 0;
       long numbUCol = 0;
       long numbICol = 0;
       long half=0;
       long temp = 0;

       while ((line = reader1.read())!=-1) {
           i++;

           if (i == 1)
           {
               temp = 0;
               temp = reader1.read();
               type += (temp*0x100) + line;
               i++;
           }
```

```
        if (i == 2)
        {
            size = readLong(reader1);
            i = i + 4;
        }

        if (i == 6)
        {
            res1 = readShort(reader1);
            i = i + 2;
        }

        if (i == 8)
        {
            res2 = readShort(reader1);
            i = i + 2;
        }

        if (i == 10)
        {
            offset = readLong(reader1);
            i = i + 4;
        }

        if (i == 14)
        {
            header = readLong(reader1);
            i = i + 4;
        }

        if (i == 18)
        {
            width = readLong(reader1);
            i = i + 4;

            height = readLong(reader1);
            i = i + 4;

            half = width;
            if((half % 2) != 0)
                half++;
            half /= 2;

            if((half % 4) != 0)
                half = (half / 4) * 4 + 4;
        }

        if (i == 26)
        {
            numbPanel = readShort(reader1);
```

```
                i = i + 2;
            }

            if (i == 28)
            {
                bitCount = readShort(reader1);
                i = i + 2;
            }

            if (i == 30)
            {
                compr = readLong(reader1);
                i = i + 4;
            }

            if (i == 34)
            {
                sCompIm = readLong(reader1);
                i = i + 4;
            }

            if (i == 38)
            {
                hRes = readLong(reader1);
                i = i + 4;
            }

            if (i == 42)
            {
                vRes = readLong(reader1);
                i = i + 4;
            }

            if (i == 46)
            {
                numbUCol = readLong(reader1);
                i = i + 4;
            }

            if (i == 50)
            {
                numbICol= readLong(reader1);
                i = i + 4;
            }

            hbi.setValues(type, size, res1, res2, offset, header,
width,
                    height, numbPanel, bitCount, compr, sCompIm,
hRes,
                    vRes, numbUCol, numbICol, half);
```

```
            if (i == offset)
            {
                reader1.mark(1);
                break;
            }
        }
        reader1.reset();

        BufferedOutputStream writer = new BufferedOutputStream (new
FileOutputStream("pixels.txt"));

        while ((line = reader1.read()) != -1)
        {
            writer.write(line);
        }
        writer.close();

        this.pr = new PrintingImage(hbi);

        return hbi;
    }

    private short readShort(BufferedInputStream reader1) throws
IOException
    {
        long temp = 0;
        short valueToreturn = 0;

        for(long j = 0x1; j <= 0x1000; j *= 0x100)
        {
            temp = reader1.read();
            valueToreturn += (temp * j);
        }
        return valueToreturn;
    }

    private long readLong(BufferedInputStream reader1) throws
IOException
    {
        long temp = 0;
        long valueToreturn = 0;

        for(long j = 0x1; j <= 0x1000000; j *= 0x100)
        {
            temp = reader1.read();
            valueToreturn += (temp * j);
        }
        return valueToreturn;
    }
}
```

Результат роботи програми