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”

Варіант 7

Виконав
студент 3 курсу
групи КП-81
Каснер Максим

Київ 2021

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

Варіант 7 :

Код програми

HeaderBitmapImage.java

```
package sample;


public class HeaderBitmapImage {
    private short type; // тип зображення або сигнатура
    private long size; // розмір файлу
    private short reserveField1; // резервоване поле №1
    private short reserveField2; // резервоване поле №2
    private long offset; // зміщення
    private long sizeOfHeader; // розмір заголовку
    private long width; // ширина
    private long height; // висота
    private short numberOfColorPlanes; // площини
    private short bitsCount; // кількість біт
    private long compression; // тип ущільнення
    private long sizeOfCompImage; // розмір ущільненого зображення
    private long horizontalResolution; // горизонтальна роздільна здатність
    private long verticalResolution; // вертикальна роздільна здатність
    private long numbOfUsedColors; // кількість кольорів палітри
```

```
    private long numbOfImportantColors; // кількість важливих кольорів
    private long halfOfWidth; // половина від ширини зображення (не міститься в заголовку)

    public HeaderBitmapImage() {
    }

    // метод для встановлення початкових значень полів класу HeaderBitmapImage
    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);
    }

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

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

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

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

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

    public void setHorizontalResolution(long horizontalResolution) {
        this.horizontalResolution = horizontalResolution;
    }

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

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

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

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

    public void setHeight(long height) {
        this.height = height;
    }

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

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

    public void setOffset(long offs) {
        offset = offs;
    }

    public void setReserveField2(short resF2) {
        reserveField2 = resF2;
    }

    public void setReserveField1(short resF1) {
        reserveField1 = resF1;
    }
```

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


    public short getType() {
        return type;
    }

    public long getHalfOfWidth() {
        return halfOfWidth;
    }

    public long getNumbOfImportantColors() {
        return numbOfImportantColors;
    }

    public long getNumbOfUsedColors() {
        return numbOfUsedColors;
    }

    public long getVerticalResolution() {
        return verticalResolution;
    }

    public long getHorizontalResolution() {
        return horizontalResolution;
    }

    public long getSizeOfCompImage() {
        return sizeOfCompImage;
    }

    public long getCompression() {
        return compression;
    }

    public short getBitsCount() {
        return bitsCount;
    }

    public short getNumberOfColorPlanes() {
        return numberOfColorPlanes;
    }

    public long getHeight() {
```

```
        return height;
    }

    public long getWidth() {
        return width;
    }

    public long getSizeOfHeader() {
        return sizeOfHeader;
    }

    public long getOffset() {
        return offset;
    }

    public short getReserveField2() {
        return reserveField2;
    }

    public short getReserveField1() {
        return reserveField1;
    }

    public long getSize() {
        return size;
    }
}
```

PrintingImage.java

```
package sample;

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

import javafx.animation.*;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.*;
import javafx.scene.shape.*;
import javafx.stage.Stage;
import javafx.util.Duration;
```

```
public class PrintingImage extends Application {
    private HeaderBitmapImage image; // приватне поле, яке зберігає об'єкт з інформацією про заголовок зображення
    private int numberOfPixels; // приватне поле для збереження кількості пікселів з чорним кольором

    public PrintingImage() {
    }

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

    @Override
    public void start(Stage primaryStage) throws Exception {
        ReadingImageFromFile.loadBitmapImage("files/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);
        int let;
        int let1;
        int let2;
        char[][] map = new char[width][height];

        // виконуємо зчитування даних про пікселі
        BufferedInputStream reader = new BufferedInputStream(new FileInputStream("files/pixels.txt"));
        for (int i = 0; i < height; i++) // поки не кінець зображення по висоті
        {
            for (int j = 0; j < half; j++) // поки не кінець зображення по довжині
            {
                let = reader.read(); // зчитуємо один символ з файлу
                let1 = let;
                let2 = let;
                let1 = let1 & (0xf0); // старший байт - перший піксель
                let1 = let1 >> 4; // зсув на 4 розряди
                let2 = let2 & (0x0f); // молодший байт - другий піксель
                if (j * 2 < width) // так як 1 символ кодує 2 пікселі нам необхідно пройти до середини ширини зображення
                {
                    if (returnPixelColor(let1).equals("BLACK")) // якщо колір пікселя чорний, то ставимо в масиві 1
```

```
                    {
                        map[j * 2][height - 1 - i] = '1';
                        numberOfPixels++; // збільшуємо кількість чорних пікселів
                    } else {
                        map[j * 2][height - 1 - i] = '0';
                    }
                }
                if (j * 2 + 1 < width) // для другого пікселя
                {
                    if (returnPixelColor(let2).equals("BLACK")) {
                        map[j * 2 + 1][height - 1 - i] = '1';
                        numberOfPixels++;
                    } else {
                        map[j * 2 + 1][height - 1 - i] = '0';
                    }
                }
            }
        }
        primaryStage.setTitle("Lab3");
        primaryStage.setScene(scene); // ініціалізуємо сцену
        primaryStage.show(); // візуалізуємо сцену
        reader.close();

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

        BufferedOutputStream writer = new BufferedOutputStream(new
FileOutputStream("files/map.txt")); // записуємо карту для руху по траекторії в файл
        for (int i = 0; i < height; i++) // поки не кінець зображення по висоті
        {
            for (int j = 0; j < width; j++) // поки не кінець зображення по довжині
            {
                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);

        // далі необхідно зробити рух об'єкту по заданій траеторії
        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])
            );
        }

        // малюнок
        Color redEdge = Color.rgb(172, 19, 24);
        Color red = Color.rgb(221, 23, 32);
        Color lightRedLine = Color.rgb(220, 78, 63);
        Color brown = Color.rgb(114, 57, 4);
        Color lightPink = Color.rgb(251, 219, 206);

        LinearGradient colorInsideHeart = new LinearGradient(65, 85, 145, 166,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, Color.rgb(245, 137, 108)), new Stop(1, Color.rgb(234, 29, 36)));

        LinearGradient colorInsideTape = new LinearGradient(0, 170, 75, 170,
                false, CycleMethod.REFLECT,
                new Stop(0, Color.rgb(152, 92, 38)), new Stop(1, Color.rgb(253, 243, 171)));

        Color repaint = Color.rgb(234, 29, 36);
        LinearGradient colorInsideSmallEdge = new LinearGradient(109, 232, 137, 280,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, repaint), new Stop(1, Color.rgb(248, 135, 103)));
        LinearGradient colorInsideSmallTriangle = new LinearGradient(0, 260, 0, 285,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, repaint), new Stop(1, Color.rgb(252, 107, 80)));

        // ліва задня часть стрічки
        {
            Path path = new Path();
            path.getElements().addAll(
                    new MoveTo(40, 158),
                    //new ArcTo(30, 10, 0, 0, 142, false, false)
                    //new ArcTo(30, 10, 0, 232, 132, false, true)
                    new ArcTo(24, 10, 0, 34, 128, false, true)
            );
            path.setStroke(brown);
            path.setFill(brown);
            root.getChildren().add(path);
        }
        // права задня часть стрічки
        {
            Path path = new Path();
            path.getElements().addAll(
```

```
                new MoveTo(240, 176),
                new ArcTo(30, 10, 15, 287, 195, false, true),
                new ArcTo(30, 10, 0, 230, 195, false, true)
                //new ArcTo(30, 12, 10, 231, 195, true, true));
        );
        path.setStroke(brown);
        path.setFill(brown);
        root.getChildren().add(path);
    }

    // 'грубе' серце
    {
        Path path = new Path();
        path.getElements().addAll(
                // left upper heart hide
                new MoveTo(47, 160),
                new ArcTo(50, 60, 0, 151, 99, true, true),
                // right upper heart hide
                new ArcTo(45, 45, 0, 251, 125, false, true),

                new QuadCurveTo(230, 220, 141, 273),
                new QuadCurveTo(47, 160, 47, 160)
        );
        path.setStrokeWidth(13);
        path.setStrokeLineJoin(StrokeLineJoin.BEVEL);
        path.setStrokeLineCap(StrokeLineCap.BUTT);
        path.setStroke(redEdge);
        path.setFill(colorInsideHeart);
        root.getChildren().add(path);
    }
    // нижня частина серця (перекраска границь)
    {
        Path path = new Path();
        path.getElements().addAll(
                new MoveTo(251, 125),
                new QuadCurveTo(230, 220, 141, 273),
                new QuadCurveTo(47, 160, 47, 160)
        );
        path.setStrokeWidth(13.2);
        path.setStrokeLineJoin(StrokeLineJoin.BEVEL);
        path.setStrokeLineCap(StrokeLineCap.BUTT);
        path.setStroke(Color.rgb(234, 29, 36));
        root.getChildren().add(path);
    }
    // нижня частина серця
    {
        Path p1 = new Path();
```

```
            p1.getElements().addAll(
                    new MoveTo(112, 230),
                    new QuadCurveTo(146, 260, 134, 285),
                    new QuadCurveTo(138, 276, 106, 240),
                    new LineTo(112, 230)
            );
            p1.setStroke(colorInsideSmallEdge);
            p1.setFill(colorInsideSmallEdge);

            Path p2 = new Path();
            p2.getElements().addAll(
                    new MoveTo(127, 250),
                    new LineTo(184, 250),
                    new QuadCurveTo(136, 287, 134, 285),
                    new QuadCurveTo(140, 255, 127, 250)
            );
            p2.setStroke(colorInsideSmallTriangle);
            p2.setFill(colorInsideSmallTriangle);
            root.getChildren().addAll(p2, p1);
        }

        // крива всередині серця
        {
            Path p = new Path();
            int startX = 144;
            int startY = 98;
            p.getElements().addAll(
                    new MoveTo(startX, startY),
                    new LineTo(158, startY - 4),
                    new QuadCurveTo(172, 100, 180, startY - 4),
                    new QuadCurveTo(168, 105, 154, 105),
                    new QuadCurveTo(146, 105, startX, startY)
            );
            p.setStroke(redEdge);
            p.setFill(redEdge);
            root.getChildren().add(p);
        }
        // міні полоски над серцем
        {
            Path p1 = new Path();
            p1.getElements().addAll(
                    new MoveTo(160, 102),
                    new QuadCurveTo(156, 102, 148, 81),
                    new QuadCurveTo(160, 60, 197, 55),
                    new QuadCurveTo(142, 80, 160, 102)
            );
            p1.setStroke(red);
```

```
            p1.setFill(red);

            Path p2 = new Path();
            p2.getElements().addAll(
                    new MoveTo(148, 81),
                    new LineTo(145, 77),
                    new QuadCurveTo(160, 55, 197, 55),
                    new QuadCurveTo(142, 80, 148, 81)
            );
            p2.setStroke(lightRedLine);
            p2.setFill(lightRedLine);
            root.getChildren().addAll(p2, p1);
        }
        // "блік" в серці
        {
            Path p = new Path();
            p.getElements().addAll(
                    new MoveTo(54, 120),
                    new QuadCurveTo(64, 67, 120, 77),
                    new QuadCurveTo(74, 77, 54, 120)
            );
            p.setStroke(lightPink);
            p.setFill(lightPink);
            root.getChildren().add(p);
        }

        // нижнє поранення
        {
            Path p = new Path();
            p.getElements().addAll(
                    new MoveTo(211, 88),
                    new QuadCurveTo(219, 89, 226, 97),
                    new QuadCurveTo(232, 106, 231, 113),
                    new QuadCurveTo(224, 98, 211, 88)
            );
            p.setStroke(Color.rgb(157, 2, 10));
            p.setFill(Color.rgb(157, 2, 10));
            root.getChildren().add(p);
        }

        // задня частина стріли
        {
            Path p1 = new Path();
            p1.getElements().addAll(
                    new MoveTo(251, 71),
                    new CubicCurveTo(260, 70, 242, 37, 251, 35),
                    new LineTo(295, 1),
```

```
                new CubicCurveTo(292, 8, 303, 23, 300, 34),
                new LineTo(251, 71)
        );
        LinearGradient color = new LinearGradient(293, 8, 268, 56,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, Color.rgb(160, 116, 55)), new Stop(1, Color.rgb(254, 204, 93)));
        p1.setStroke(color);
        p1.setFill(color);

        Path p2 = new Path();
        p2.getElements().addAll(
                new MoveTo(254, 78),
                new CubicCurveTo(259, 74, 284, 90, 290, 85),
                new LineTo(334, 52),
                new CubicCurveTo(330, 55, 306, 39, 303, 41),
                new LineTo(254, 78)
        );
        color = new LinearGradient(305, 40, 290, 73,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, Color.rgb(160, 116, 55)), new Stop(1, Color.rgb(254, 204, 93)));
        p2.setStroke(color);
        p2.setFill(color);

        Path p3 = new Path();
        p3.getElements().addAll(
                new MoveTo(225, 104),
                new LineTo(311, 34),
                new QuadCurveTo(308, 30, 306, 28),
                new LineTo(219, 97),
                new LineTo(225, 104)
        );
        color = new LinearGradient(244, 76, 251, 84,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, Color.rgb(255, 211, 143)), new Stop(1, Color.rgb(160, 105, 25)));
        p3.setStroke(color);
        p3.setFill(color);

        root.getChildren().addAll(p1, p2, p3);
    }

    // слід поранення (біля задньої частини стріли)
    {
        Path p = new Path();
        p.getElements().addAll(
                new MoveTo(231, 113),
                new QuadCurveTo(224, 98, 211, 88)
        );
```

```
            p.setStroke(Color.rgb(255, 107, 94));
            root.getChildren().add(p);
        }

        // серединя частини стріли
        {
            Path p = new Path();
            p.getElements().addAll(
                    new MoveTo(225, 104),
                    new LineTo(160, 158),
                    new LineTo(152, 153),
                    new LineTo(219, 97),
                    new LineTo(225, 104)
            );
            LinearGradient color = new LinearGradient(225, 104, 208, 109,
                    false, CycleMethod.NO_CYCLE,
                    new Stop(0, Color.rgb(240, 95, 66)), new Stop(1, Color.rgb(227, 53, 42)));
            p.setStroke(color);
            p.setFill(color);

            root.getChildren().addAll(p);
        }

        // нижнє поранення
        {
            Path p = new Path();
            p.getElements().addAll(
                    new MoveTo(92, 213),
                    new LineTo(101, 223),
                    new LineTo(89, 216),
                    new LineTo(92, 213)
            );
            p.setStroke(Color.rgb(157, 2, 10));
            p.setFill(Color.rgb(157, 2, 10));
            root.getChildren().add(p);
        }

        // кінець стріли
        {
            // кінець передньої частини стріли
            Path p = new Path();
            p.getElements().addAll(
                    new MoveTo(92, 214),
                    new LineTo(60, 237),
                    new LineTo(55, 230),
                    new LineTo(85, 207),
                    new LineTo(92, 214)
```

```
        );
        LinearGradient color = new LinearGradient(62, 224, 65, 227,
                false, CycleMethod.REFLECT,
                new Stop(0, Color.rgb(191, 145, 67)), new Stop(1, Color.rgb(243, 194, 99)));
        p.setStroke(color);
        p.setFill(color);

        // коло
        Circle circle = new Circle(53, 238, 10);
        RadialGradient circleColor = new RadialGradient(0,
                .1, 48, 233, 12, false, CycleMethod.NO_CYCLE,
                new Stop(0, Color.rgb(255, 234, 160)), new Stop(1, Color.rgb(162, 92, 7)));
        circle.setFill(circleColor);

        // серцевина стріли
        Path endInside = new Path();
        endInside.getElements().addAll(
                new MoveTo(47, 235),
                new QuadCurveTo(27, 218, 13, 271),
                new QuadCurveTo(66, 266, 54, 245),
                new LineTo(47, 235)
        );
        color = new LinearGradient(27, 242, 40, 256,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, Color.rgb(198, 133, 53)), new Stop(1, Color.rgb(255, 221, 137)));
        endInside.setStroke(color);
        endInside.setFill(color);

        // верхня зовнішня сторона стріли
        Path endOutside1 = new Path();
        endOutside1.getElements().addAll(
                new MoveTo(53, 236),
                new QuadCurveTo(20, 200, 2, 280),
                new LineTo(53, 236)
        );
        color = new LinearGradient(20, 230, 27, 238,
                false, CycleMethod.NO_CYCLE,
                new Stop(0, Color.rgb(190, 149, 59)), new Stop(1, Color.rgb(244, 207, 136)));
        endOutside1.setStroke(color);
        endOutside1.setFill(color);

        // нижня зовнішня сторона стріли
        Path endOutside2 = new Path();
        endOutside2.getElements().addAll(
                new MoveTo(2, 280),
                new QuadCurveTo(90, 274, 53, 236),
                new LineTo(2, 280)
```

```
            );
            color = new LinearGradient(36, 251, 48, 270,
                    false, CycleMethod.NO_CYCLE,
                    new Stop(0, Color.rgb(168, 91, 9)), new Stop(1, Color.rgb(255, 221, 137)));
            endOutside2.setStroke(color);
            endOutside2.setFill(color);

            root.getChildren().addAll(endOutside1, endOutside2, endInside, circle, p);
        }

        // стрічка
        {
            Path p = new Path();
            p.getElements().addAll(
                    new MoveTo(0, 142),
                    new CubicCurveTo(85, 202, 197, 72, 287, 142),
                    new LineTo(287, 192),
                    new CubicCurveTo(197, 132, 85, 252, 0, 192),
                    new LineTo(0, 192)
            );
            p.setStroke(colorInsideTape);
            p.setFill(colorInsideTape);
            root.getChildren().add(p);
        }

        //animation

        int cycleCount = 2;
        int time = 4000;

        PathTransition pathTransition = new PathTransition();
        pathTransition.setDuration(Duration.millis(time));
        pathTransition.setPath(path2);
        pathTransition.setNode(root);

        RotateTransition rotateTransition = new RotateTransition(Duration.millis(time), root);
        rotateTransition.setByAngle(360);
        rotateTransition.setCycleCount(cycleCount);
        rotateTransition.setAutoReverse(true);

        ScaleTransition scaleTransition = new ScaleTransition(Duration.millis(time), root);
        scaleTransition.setToX(0.2);
        scaleTransition.setToY(0.2);
        scaleTransition.setCycleCount(cycleCount);
        scaleTransition.setAutoReverse(true);

        ParallelTransition parallelTransition = new ParallelTransition();
```

```
        parallelTransition.getChildren().addAll(
                rotateTransition,
                scaleTransition,
                pathTransition
        );
        parallelTransition.setCycleCount(Timeline.INDEFINITE);
        parallelTransition.play();


    }

    private String returnPixelColor(int color) // метод для співставлення кольорів 16-бітного
зображення
    {
        String col = "BLACK";
        return switch (color) {
            case 0 -> "BLACK";
            case 1 -> "LIGHTCORAL";
            case 2 -> "GREEN";
            case 3 -> "BROWN";
            case 4 -> "BLUE";
            case 5 -> "MAGENTA";
            case 6 -> "CYAN";
            case 7 -> "LIGHTGRAY";
            case 8 -> "DARKGRAY";
            case 9 -> "RED";
            case 10 -> "LIGHTGREEN";
            case 11 -> "YELLOW";
            case 12 -> "LIGHTBLUE";
            case 13 -> "LIGHTPINK";
            case 14 -> "LIGHTCYAN";
            case 15 -> "WHITE";
            default -> col;
        };
    }

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

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() {

    }


    // метод, який приймає потік даних зчитаних із файлу із зображенням

    // та повертатиме об’єкт типу HeaderBitmapImage, з інформацією про зображення

    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;
        while ((line = reader1.read()) != -1) { // поки не кінець файлу
            i++; // збільшуємо лічильник кількості байт заголовку зображення
            if (i == 1) // зчитуємо сигнатуру
            {
                temp = reader1.read();
                type += (temp * 0x100) + line;
                i++;
            }
            if (i == 2) // якщо зміщення відносно початку файлу = 2, то зчитуємо розмір файлу
            {
                size = readLong(reader1); // у змінну size записуємо результат роботи методу
readLong
                i = i + 4; // додаємо 4 до кількості зчитаних байт з файлу, так як розмір поля
size 4 байти
            }
            if (i == 6) //зчитуємо резервоване поле №1
            {
                res1 = readShort(reader1); // у змінну res1 записуємо результат роботи методу
readShort
                i = i + 2; // додаємо 2 до кількості зчитаних байт з файлу, так як розмір поля
res1 2 байти
            }
            if (i == 8) //зчитуємо резервоване поле №2
            {
                res2 = readShort(reader1);
                i = i + 2;
            }
            if (i == 10) //зчитуємо зміщення
            {
                offset = readLong(reader1);
                i = i + 4;
            }
            if (i == 14) //зчитуємо розмір заголовку
            {
                header = readLong(reader1);
```

```
                i = i + 4;

            }
// зчитуємо з 18ої та 22ої позиції ширину і довжину зображення
            if (i == 18) {

                width = readLong(reader1);

                i = i + 4;

                height = readLong(reader1);

                i = i + 4;

                half = width;

                if ((half % 2) != 0) // перевірка чи ширина зображення кратна 2 і якщо ні, то
збільшуємо це значення на 1

                    half++; // щоб доповнити значення половини від ширини зображення

                half /= 2;

                if ((half % 4) != 0) // якщо не ділиться на 4

                    half = (half / 4) * 4 + 4; // доповнюємо значення половини ширини
зображення, щоб вона була кратна 4

            }

            if (i == 26) //зчитуємо кількість площин

            {

                numbPanel = readShort(reader1);

                i = i + 2;

            }

            if (i == 28) //зчитуємо кількість біт

            {

                bitCount = readShort(reader1);

                i = i + 2;

            }

            if (i == 30) //зчитуємо тип ущільнення

            {

                compr = readLong(reader1);

                i = i + 4;

            }

            if (i == 34) //зчитуємо розмір ущільненого зображення

            {

                sCompIm = readLong(reader1);

                i = i + 4;

            }
```

```
            if (i == 38) // горизонтальна роздільна здатність
            {
                hRes = readLong(reader1);
                i = i + 4;
            }
            if (i == 42) // вертикальна роздільна здатність
            {
                vRes = readLong(reader1);
                i = i + 4;
            }
            if (i == 46) // кількість кольорів палітри
            {
                numbUCol = readLong(reader1);
                i = i + 4;
            }
            if (i == 50) // кількість важливих кольорів
            {
                numbICol = readLong(reader1);
                i = i + 4;
            }
// записуємо усі зчитані значення в об'єкт
            hbi.setValues(type, size, res1, res2, offset, header, width,
                    height, numbPanel, bitCount, compr, sCompIm, hRes,
                    vRes, numbUCol, numbICol, half);
// помічяємо місце в потоці, де починаються пікселі
            if (i == offset) {
                reader1.mark(1);
                break;
            }
        }
// вертаємося на те місце звідки мають починатися пікселі
        reader1.reset();
// запишемо в окремий файл частину зображення, з якої починаються власне пікселі
        BufferedOutputStream writer = new BufferedOutputStream(new
FileOutputStream("files/pixels.txt"));
        while ((line = reader1.read()) != -1) {
```

```
            writer.write(line);

        }

        writer.close();

        this.pr = new PrintingImage(hbi);

        return hbi;

    }


    // метод для коректного читання полів розміром 2 байти у форматі запису little-endian

// та переводу в десяткову систему числення

    private short readShort(BufferedInputStream reader1) throws IOException {

        long temp;

        short valueToreturn = 0;

        for (long j = 0x1; j <= 0x1000; j *= 0x100) // цикл від 1 до 8 з кроком j*4 -
відповідно кількість виконань циклу = 2

        {

            temp = reader1.read();

            valueToreturn += (temp * j); // додаємо до поточного числа значення нового розряду
записаного у 10-вій системі числення

        }

        return valueToreturn;

    }


    // метод для коректного читання полів розміром 4 байти у форматі запису little-endian

// та переводу в десяткову систему числення

    private long readLong(BufferedInputStream reader1) throws IOException {

        long temp;

        long valueToreturn = 0;

        for (long j = 0x1; j <= 0x1000000; j *= 0x100) // цикл від 1 до 64 з кроком j*4 -
відповідно кількість виконань циклу = 4

        {

            temp = reader1.read();

            valueToreturn += (temp * j); // додаємо до поточного числа значення нового розряду
записаного у 10-вій системі числення

        }

        return valueToreturn;

    }

}
```

ReadingImageFromFile.java

```
package sample;


import java.io.BufferedInputStream;

import java.io.BufferedOutputStream;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;


public class ReadingImageFromFile {

    public static PrintingImage pr;


    // метод для читання файлу з зображенням та виведення інформації про зчитаний файл,

// який приймає на вхід назву файлу з зображенням у форматі BMP

    public static void loadBitmapImage(String filename) throws IOException {

        int line;

        BufferedInputStream reader = new BufferedInputStream(new FileInputStream(filename)); //
потік для читання

        BufferedOutputStream writer = new BufferedOutputStream(new
FileOutputStream("files/primer_bmp.txt")); // потік для запису зчитаних зображень про пікселі у
ASCII кодах

        while ((line = reader.read()) != -1) {

            writer.write(line);

        }

        reader.close();

        writer.close();

        BufferedInputStream reader1 = new BufferedInputStream(new
FileInputStream("files/primer_bmp.txt")); // потік

        ReadingHeaderFromBitmapImage reading = new ReadingHeaderFromBitmapImage(); //створення
об’єкту типу ReadingHeaderFromBitmapImage

        HeaderBitmapImage hbi = reading.Reading(reader1); // у об'єкт типу HeaderBitmapImage
записуємо результат роботи методу читання заголовку файлу

        pr = reading.pr;

// блок виведення зчитаної інформації на консоль

        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) {
    }
}
```

Результати роботи програми :