

## Лабораторна робота №3

**Тема:** Структура файлів формату *.bmp*. Анімація примітивів за допомогою засобів бібліотеки *JavaFX*

**Мета:**

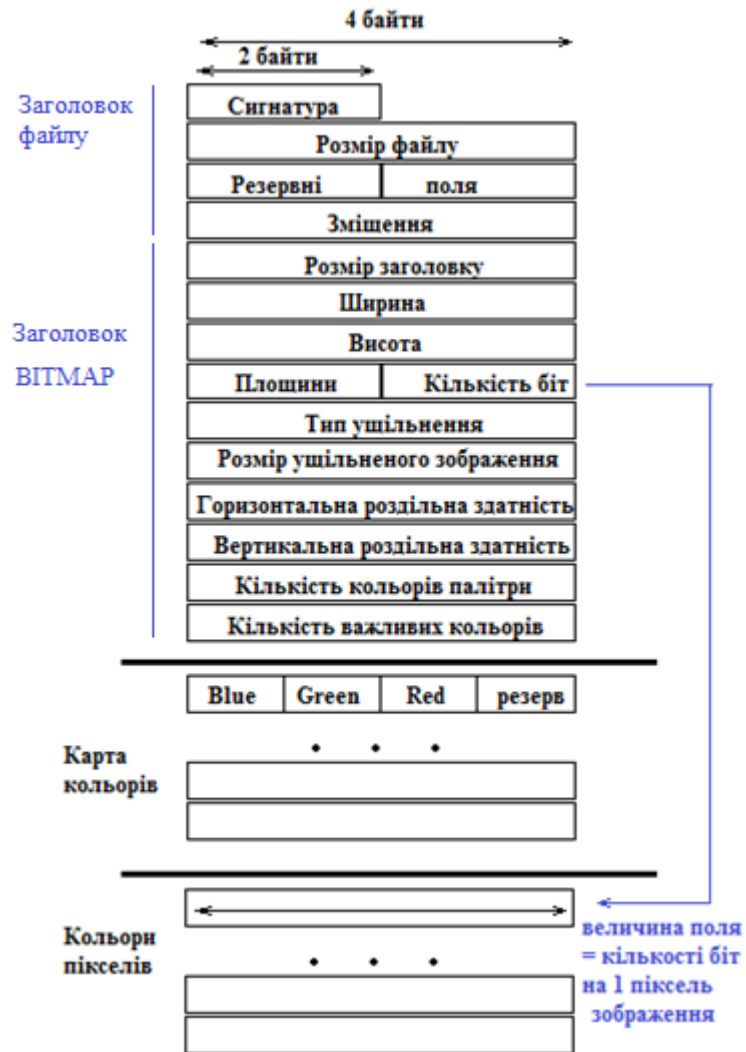
- 1) вивчення структури та особливостей використання файлів формату *.bmp*;
- 2) вивчення стандартних засобів *JavaFX* для візуалізації зображення;
- 3) вивчення засобів анімації примітивів в *JavaFX*.

### Теоретичні відомості

#### Структура файлу-зображення з розширенням *.BMP*

**BMP** або *Bitmap Picture* – це формат файлів для збереження растрових двовимірних зображень, розроблений компанією *Microsoft*.

Файл формату *.bmp* складається з 3 частин: заголовку файлу, заголовку *BITMAP* та основної частини. Заголовки завжди мають фіксований розмір, а основна частина – змінний розмір в залежності від кількості пікселів зображення та кольорової гама, що використовується. Основна частина складається з палітри кольорів (може біти відсутня) та самого зображення. Розглянемо докладніше структуру файлу з розширенням *.bmp* (Рис. 2).



**Рис. 1. Структура файлу формату .BMP**

При розробці програм, в яких відбувається зчитування даних зображень у форматі *.bmp* необхідно враховувати, що порядок байтів у файлах такого типу – *little-endian* (Рис. 3).

<b>little-endian</b>				
76 <sub>A0</sub>	00 <sub>A1</sub>	00 <sub>A2</sub>	00 <sub>A3</sub>	$00000076_{16} = 118_{10}$
<b>big-endian</b>				
76 <sub>A3</sub>	00 <sub>A2</sub>	00 <sub>A1</sub>	00 <sub>A0</sub>	$76000000_{16} = 123731968_{10}$
<b>little-endian</b>				
F6 <sub>A0</sub>	A9 <sub>A1</sub>	03 <sub>A2</sub>	00 <sub>A3</sub>	$0003A9F6_{16} = 240118_{10}$
<b>big-endian</b>				
00 <sub>A3</sub>	03 <sub>A2</sub>	A9 <sub>A1</sub>	F6 <sub>A0</sub>	$0003A9F6_{16} = 240118_{10}$

**Рис. 2. Порядок байтів**

Розглянемо структуру файлу за допомогою таблиці 1, в якій представлено найбільш популярну структуру файлів формату *.bmp*.

**Таблиця 1. Поля структури BMP файлу**

Поле	Пояснення	Зміщення (байт)	Розмір поля (байт)
Сигнатура	Тип файлу. Для файлів формату <i>.bmp</i> в цьому полі завжди “BM” (в HEX: 0x42 0x4d).	0	2
Розмір файлу	Розмір всього файлу в байтах.	2	4
Резервне поле 1	Поле зарезервоване і для файлів формату <i>.bmp</i> має містити 0.	6	2
Резервне поле 2	Поле зарезервоване і для файлів формату <i>.bmp</i> має містити 0.	8	2
Зміщення	Містить зміщення в байтах від початку файлу до початку послідовності кольорів пікселів (з якого починається саме зображення).	10	4
Розмір заголовку	Розмір заголовку <i>BITMAP</i> (в байтах). Для файлів формату <i>.bmp</i> дорівнює завжди 40.	14	4
Ширина	Ширина зображення в пікселях.	18	4
Висота	Висота зображення в пікселях	22	4
Площини	Число кольорових площин і в форматі <i>.bmp</i> завжди містить 1.	26	2
Кількість біт	Кількість біт на піксель. Може приймати значення 0,1,2,4,8,16,24,32. 0 – використовувався в Win98/Me/2000/XP. 1 – зображення монохромне. Кожен біт зображення представляє 1 піксель. 2 – не використовується, адже не відповідає жодному зі стандартів. 4 – шістнадцять зображення. Колір пікселя визначається 4-х бітними індексами, кожен байт зображення відповідає за колір двох пікселів. 8 – в палітрі 256 кольорів, кожен байт зображення = 1 пікселю. 16 – кожні 2 байти зображення зберігають інформацію про колір 1 пікселя. 24 – палітра не використовується, а кожні 3 байти зображення представляють один піксель (по 1 байту для синього, зеленого та червоно каналу). 32 – кожні 4 байти зображення представляють один піксель (по 1 байту для інтенсивності, синього, зеленого та червоного каналу відповідно)	28	2
Тип ущільнення	Тип ущільнення зображення. Може приймати значення: 0,1,2,3,4,5,6. 0 – не використовувалося ущільнення. 1 – ущільнення RLE для 8-бітних зображень. 2 – ущільнення RLE для 4-бітних зображень.	30	4

	3 – зображення не ущільнене, але містить 3 маски для компонент кольору. Використовується в 16- та 32-бітних зображеннях. 4 – JPEG-ущільнення. 5 – PNG-ущільнення. 6 – зображення не ущільнене, але містить 4 маски для компонент кольору. Використовується в 16- та 32-бітних зображеннях.		
Розмір ущільненого зображення	Розмір ущільненого зображення в байтах.	34	4
Горизонтальна роздільна здатність	Горизонтальна роздільна здатність (піксель/метр) для цільового пристрою .	38	4
Вертикальна роздільна здатність	Вертикальна роздільна здатність (піксель/метр) для цільового пристрою .	42	4
Кількість кольорів палітри	Кількість кольорових індексів, які використовуються в палітрі. Дорівнює 0, якщо зображення використовує максимально дозволена кількість індексів.	46	4
Кількість важливих кольорів	Кількість елементів палітри, які необхідні для відображення зображення. Якщо містить 0 – то всі індекси важливі.	50	
Карта кольорів	Може містити послідовність 4-байтових полів по числу доступних кольорів. Три молодших байти визначають інтенсивність червоної, зеленої та синьої компонент кольору, а старший байт не використовується. Файл може не містити палітри, якщо в ньому зберігається не ущільнене зображення.	54	4*N
Кольори пікселів	Послідовність пікселів, записаних у тому чи іншому вигляді. Пікселі зберігаються порядково знизу вверх (Нижній правий піксель зображення у файлі буде першим). Кожний рядок зображення доповнюється нулями до довжини, що кратна 4 байтам.	54+4*N	(Розмір файлу) – (Зміщення)

Створимо проект під назвою *Bitmap\_lab3*, підключимо бібліотеку *JavaFX* та в ньому створимо пакет *lab3*, в якому знаходитимуться класи для читання структури файлу формату *.bmp*, збереження інформації про зображення та візуалізації даного зображення за допомогою примітивів *JavaFX*. Також необхідно створити папку, в якій зберігатимуться додаткові файли (наприклад, зображення). Для цього необхідно натиснути правою кнопкою миші по назві проекту *New* → *Source Folder*.

Створимо клас *HeaderBitmapImage*, який відповідатиме за збереження інформації, що міститься в заголовках файлу із зображенням.

```
package lab3;
```

```

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 () {} // конструктор

    public void setType (short type) // метод "сеттер" для безпечного встановлення
    значення поля type
    {
        this.type = type;
    }

    public short getType () // метод "геттер" для безпечного отримання значення, що
    зберігається в полі type
    {
        return type;
    }

    // *****

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

```

```

        setNumOfUsedColors(nUsCol);
        setNumOfImportantColors(nImpCol);
        setHalfOfWidth(half);
    }
}

```

Створимо клас *ReadingHeaderFromBitmapImage*, який відповідатиме за коректне зчитування інформації з файлу формату *.bmp*. В даному класі нам необхідні будуть методи *readShort* та *readLong* для коректного читання полів розміром 2 байти та 4 байти у форматі запису *little-endian* та переводу в десяткову систему числення. Крім того, необхідний метод, який дозволить коректно читати та записувати потрібну інформацію у поля відповідного об'єкта типу *HeaderBitmapImage*.

```

package lab3;

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
    // метод, який приймає потік даних зчитаних із файлу із зображенням
    // та повертатиме об'єкт типу HeaderBitmapImage, з інформацією про зображення
    {
        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) { // поки не кінець файлу
            i++; // збільшуємо лічильник кількості байт заголовку зображення

```

```

        if (i==1) // зчитуємо сигнатуру
        {
            temp = 0;
            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);

    // помічаємо 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("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 переведення в десяткову систему числення
    private short readShort(BufferedInputStream reader1) throws IOException
    {
        long temp = 0;
        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 переведення в десяткову систему числення
    private long readLong(BufferedInputStream reader1) throws IOException
    {
        long temp = 0;
        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;
    }
}

```

Тепер нам необхідний клас *ReadingImageFromFile*, для читання файлу з зображенням та виведення інформації про зчитаний файл. В даному класі нам необхідний метод *loadBitmapImage*, в якому вкажемо буде відбуватися зчитування необхідної інформації з зображення та її виведення на екран.

```

package lab3;

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("primer_bmp.txt")); // потік для запису зчитаних зображень про
пкселі у ASCII кодах
        while ((line = reader.read())!=-1) {
            writer.write(line);
        }
        reader.close();
        writer.close();

        BufferedInputStream reader1 = new BufferedInputStream (new
FileInputStream("primer_bmp.txt")); // потік
        ReadingHeaderFromBitmapImage reading = new
ReadingHeaderFromBitmapImage(); //створення об'єкту типу ReadingHeaderFromBitmapImage
        HeaderBitmapImage hbi = new HeaderBitmapImage(); // створення об'єкту
типу 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.getNumOfUsedColors());
        System.out.println("number of important colors =
"+hbi.getNumOfImportantColors());

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

        reader1.close();
    }

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

```

Створимо клас *PrintingImage*, який буде виконувати візуалізацію зображення у нашій програмі. Так як ми будемо використовувати бібліотеку *JavaFX*, то клас

обов'язково має наслідуватися від класу *Application*. Для створення сцени необхідно перевизначити метод *start*, в якому ми реалізуємо відображення пікселів за допомогою кола радіусом 1 піксель та методу *returnPixelColor*, який порівнюватиме кольори *JavaFX* з кольорами у файлі з зображенням (16-біт).

```
package lab3;

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

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

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) // перевизначений стандартний
    конструктор
    {
        this.image = image;
    }

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

        ReadingImageFromFile.LoadBitmapImage("D:/Eclipse_workspace/JavaFX_Lab2_for_Com
        p_Graphics_Labs/sources/tr5.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);
        Circle cir;

        int let = 0;
        int let1 = 0;
        int let2 = 0;
        char[][] map = new char[width][height];
        // виконуємо зчитування даних про пікселі
        BufferedInputStream reader = new BufferedInputStream (new
        FileInputStream("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
        {
            cir = new Circle ((j)*2,(height-1-i),1,
Color.valueOf((returnPixelColor(let1)))); // за допомогою стандартного
// примітива Коло радіусом в 1 піксель та кольором
визначеним за допомогою методу returnPixelColor малюємо піксель
            root.getChildren().add(cir); //додаємо об'єкт в
сцену

            if (returnPixelColor(let1) == "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
        {
            cir = new Circle ((j)*2+1,(height-1-
i),1,Color.valueOf((returnPixelColor(let2))));
            root.getChildren().add(cir);
            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); // ініціалізуємо сцену
    primaryStage.show(); // візуалізуємо сцену

    reader.close();

    BufferedOutputStream writer = new BufferedOutputStream (new
FileOutputStream("map.txt")); // записуємо карту для руху по траєкторії 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
        {
            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 траєкторії
}

private String returnPixelColor (int color) // метод для співставлення
кольорів 16-бітного зображення
{
    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);
}
}

```

Результати роботи програми в консолі:

```

type = 19778
size = 80118
reserve field 1 = 0
reserve field 2 = 0
offset = 118
size of header = 40
width = 400
height = 400
number of planes = 1
number of bits = 4
type of compression = 0
size of image after compression = 0
horizontal resolution = 0
vertical resolution = 0
number of used colors = 16
number of important colors = 16
half of width = 200
number of black color pixels = 2530

```

Результати роботи програми в *Application*:

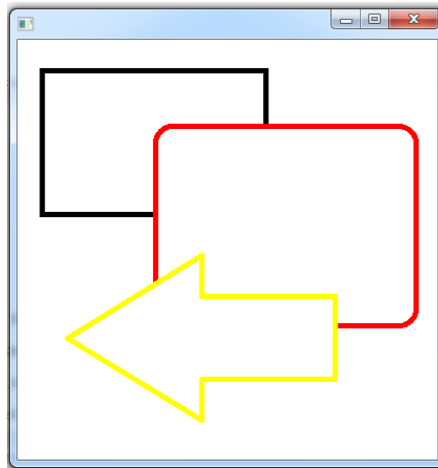


Рис. 3. Результат роботи програми

### Стандартні засоби відображення зображень в JavaFX

Бібліотека *JavaFX* має стандартні засоби для візуалізації зображень (*Image Ops API*), які на даний момент працюють із зображеннями форматів *.jpg*, *.bmp* та *.png*. Головним недоліком на даний момент є те, що монохромні, 16-бітні та 256-бітні *.bmp* зображення не можуть коректно відображатися за допомогою стандартних функцій роботи із зображеннями.

*Image Ops API* складається з таких ключових класів/інтерфейсів в пакеті *javafx.scene.image*:

- *Image* – клас, що використовується для завантаження зображення (синхронно або асинхронно), яке розташоване локально на комп'ютері, або для завантаження по *URL*.
- *ImageView* – клас, що використовується для візуалізації зображення, яке було завантажене за допомогою класу *Image*.
- *WritableImage* – клас-нащадок *Image*. Цей клас має метод *PixelWriter* для запису пікселів зображення. Конструктори класу створюють або порожній образ зображення вказаних розмірів готовий до запису в нього інформації або відображають зображення передане за допомогою *PixelReader*.
- *PixelReader* – інтерфейс, який визначає методи для отримання даних з пікселів зображення.
- *PixelWriter* – інтерфейс, який визначає методи для запису даних про пікселі зображення.
- *PixelFormat* – об'єкт, який визначає структуру даних для пікселів даного формату.
- *WritablePixelFormat* – об'єкт типу *PixelFormat*, який представляє формат пікселів, та зберігає усі кольори і тому може бути використаний в якості кінцевого формату для запису довільних повнокольорових зображень.

Розглянемо найпростіший приклад роботи з завантаженням зображень в *JavaFX*, для роботи якого в клас необхідно імпортувати:

```
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;
```

Для початку необхідно створити об'єкт класу *Image* та передати параметром шлях до файлу з зображенням або *URL*.

```
Image image = new Image("http://docs.oracle.com/javafx/" + "javafx/images/javafx-  
documentation.png");
```

Потім створюємо об'єкт *ImageView* та за допомогою методу *setImage* задаємо зображення, яке ми матимемо відображати.

```
ImageView imageView = new ImageView();  
    imageView.setImage(image);  
або  
Image image = new Image("tr55.bmp");
```

Далі після стандартної процедури створення сцени та додавання ключового об'єкта сцени *root*, додамо об'єкт зображення в сцену та візуалізуємо її.

```
// Display image on screen  
StackPane root = new StackPane();  
root.getChildren().add(imageView);  
Scene scene = new Scene(root, 300, 250);  
primaryStage.setTitle("Image Read Test");  
primaryStage.setScene(scene);  
primaryStage.show();
```

Однією з переваг використання стандартних засобів візуалізації зображення в *JavaFX* є можливість отримання інформації про складові кольору та інтенсивності для кожного пікселя повнокольорового зображення різних форматів.

Для роботи з пікселями в програмі необхідно імпортувати класи

```
import javafx.scene.image.PixelReader;  
import javafx.scene.paint.Color;
```

Після створення об'єктів *Image* та *ImageView* необхідно створити об'єкт типу *PixelReader*, який за допомогою методу *getPixelReader* дозволить отримати інформацію не тільки про кожен піксель зображення, але й про основні властивості зображення (розміри, формат пікселів, тощо).

```
PixelReader pixelReader = image.getPixelReader();  
System.out.println("Image Width: "+image.getWidth());  
System.out.println("Image Height: "+image.getHeight());  
System.out.println("Pixel Format: "+pixelReader.getPixelFormat());
```

У задачах, в яких необхідно візуалізувати частини зображень, або проводити обробку за певним критерієм необхідно вміти виділити з всього обсягу пікселів лише необхідні.

```
// Determine the color of each pixel in the image  
for (int readY = 0; readY < image.getHeight(); readY++) {  
    for (int readX = 0; readX < image.getWidth(); readX++) {  
        Color color = pixelReader.getColor(readX, readY);  
        System.out.println("\nPixel color at coordinates ("  
            + readX + ", " + readY + ") "  
            + color.toString());  
        System.out.println("R = " + color.getRed());
```

```

        System.out.println("G = " + color.getGreen());
        System.out.println("B = " + color.getBlue());
        System.out.println("Opacity = " + color.getOpacity());
        System.out.println("Saturation = " + color.getSaturation());
    }
}

```

Після запуску програми в консолі буде отримано перелік даних про кожен піксель:

```

Image Width: 159.0
Image Height: 59.0
Pixel Format: javafx.scene.image.PixelFormat$ByteRgb@9ed91f
....
Pixel color at coordinates (7,56) 0xffffffff
R = 1.0
G = 1.0
B = 1.0
Opacity = 1.0
Saturation = 0.0

Pixel color at coordinates (8,56) 0xfefeffff
R = 0.9960784316062927
G = 0.9960784316062927
B = 1.0
Opacity = 1.0
Saturation = 0.003921568393707275

Pixel color at coordinates (9,56) 0xfcfdffff
R = 0.9882352948188782
G = 0.9921568632125854
B = 0.9921568632125854
Opacity = 1.0
Saturation = 0.00395256893250661
...

```

Стандартні методи класу *WritableImage* дозволяють змінювати значення кольорів пікселів та записувати результат змін у новий файл з зображенням. Розглянемо процес редагування зображення. Для початку імпортуємо в клас:

```

import javafx.scene.image.PixelWriter;
import javafx.scene.image.WritableImage;

```

Створимо об'єкти типу *WritableImage* та *PixelWriter* для можливості запису зображення та його модифікації:

```

// Create WritableImage
WritableImage wImage = new WritableImage(
(int)image.getWidth(), (int)image.getHeight());

PixelWriter pixelWriter = wImage.getPixelWriter();

```

У циклах, які проходять по зображенню після зчитування інформації додамо наступні рядки, які дозволять змінити яскравість зображення кожного пікселя у новому зображенні:

```

// Now write a brighter color to the PixelWriter
color = color.brighter();
pixelWriter.setColor(readX, readY, color);

```



## Анімація в JavaFX

В *JavaFX* анімація представлена класом *javafx.animation*. Анімація в *JavaFX* розділяється на часову анімацію та переходи (перетворення) об'єкта.

Класи стандартних перетворень в *JavaFX*:

- 1) *FadeTransition* – ефект зникнення.
- 2) *FillTransition* – ефект заливки зображення.
- 3) *ParallelTransition* – програвання декількох анімацій паралельно.
- 4) *PathTransition* – створення анімації руху по траєкторії.
- 5) *PauseTransition* – створення паузи при анімації.
- 6) *RotateTransition* – ефект повороту.
- 7) *ScaleTransition* – ефект масштабування.
- 8) *SequentialTransition* – програвання декількох анімацій послідовно.
- 9) *StrokeTransition* – зміна кольору фігури.
- 10) *TranslateTransition* – переміщення об'єкта.

Для можливості використання будь-якого з класів анімації об'єкта, спочатку його необхідно імпортувати в програму. Наприклад,

```
import javafx.animation.FadeTransition;
```

Для можливості виконання анімації циклічно або протягом якогось певного часу також необхідно імпортувати клас:

```
import javafx.animation.Timeline;
```

Створимо примітив Прямокутник та застосуємо до нього ефект зникнення *FadeTransition*. Для створення будь-якої анімації необхідно буде створити об'єкт класу необхідної вам анімації та в параметрах вказати час виконання анімації та об'єкт, до якого застосовуватиметься ця анімація:

```
FadeTransition ft = new FadeTransition(Duration.millis(3000), rect1);
```

Будь-який вид анімації передбачає встановлення певних значень, в межах яких буде відбуватися зміна відповідного параметра. Наприклад, для анімації *FadeTransition* такими параметрами є початкове і кінцеве значення прозорості об'єкта, які встановлюються наступним чином:

```
ft.setFromValue(1.0);  
ft.setToValue(0.1);
```

Також для анімації можна вказати, що вона буде відбуватися циклічно. Для цього необхідно буде змінити значення часових характеристик анімації за допомогою методів:

```
ft.setCycleCount(Timeline.INDEFINITE);  
ft.setAutoReverse(true);
```

Для того, щоб відтворити анімацію необхідно записати наступний рядок коду:

```
ft.play();
```

Кінцевий вигляд програми для анімації прямокутника виглядає наступним чином:

```

final Rectangle rect1 = new Rectangle(10, 10, 100, 100);
rect1.setArcHeight(20);
rect1.setArcWidth(20);
rect1.setFill(Color.RED);
root.getChildren().add(rect1);
FadeTransition ft = new FadeTransition(Duration.millis(3000), rect1);
ft.setFromValue(1.0);
ft.setToValue(0.1);
ft.setCycleCount(Timeline.INDEFINITE);
ft.setAutoReverse(true);
ft.play();

```

Приклад програми, в якій виконується рух прямокутника по траєкторії:

```

// створення синього прямокутника з круглими кутами
final Rectangle rectPath = new Rectangle (0, 0, 40, 40);
rectPath.setArcHeight(10);
rectPath.setArcWidth(10);
rectPath.setFill(Color.BLUE);
root.getChildren().add(rectPath);

// створення траєкторії з 2 ліній типу CubicCurveTo
Path path = new Path();
path.getElements().add(new MoveTo(20,20)); // вказання початкової позиції, з якої
// починається траєкторія
path.getElements().add(new CubicCurveTo(380, 0, 380, 120, 200, 120)); // перша крива
path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 380, 240)); // друга крива

// створення анімації руху по траєкторії
PathTransition pathTransition = new PathTransition();
pathTransition.setDuration(Duration.millis(4000)); // встановлення часу анімації
pathTransition.setPath(path); // прив'язування траєкторії
pathTransition.setNode(rectPath); // вибір об'єкта, який буде анімуватися

pathTransition.setOrientation(PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
// вказання орієнтації об'єкта при русі
pathTransition.setCycleCount(Timeline.INDEFINITE); // циклічна анімація
pathTransition.setAutoReverse(true); // можливість руху назад
pathTransition.play(); // відтворення анімації

```

Приклад програми, в якій виконується паралельно анімація зникнення, переміщення, масштабування та повороту:

```

//Створення прямокутника
Rectangle rectParallel = new Rectangle(10,200,50, 50);
rectParallel.setArcHeight(15);
rectParallel.setArcWidth(15);
rectParallel.setFill(Color.DARKBLUE);
rectParallel.setTranslateX(50);
rectParallel.setTranslateY(75);
root.getChildren().add(rectParallel);

// створення ефекту зникнення
FadeTransition fadeTransition =
    new FadeTransition(Duration.millis(3000), rectParallel);
fadeTransition.setFromValue(1.0f);
fadeTransition.setToValue(0.3f);
fadeTransition.setCycleCount(2);
fadeTransition.setAutoReverse(true);

```

```

// Створення ефекту переміщення
TranslateTransition translateTransition =
    new TranslateTransition(Duration.millis(2000), rectParallel);
translateTransition.setFromX(50);
translateTransition.setToX(350);
translateTransition.setCycleCount(2);
translateTransition.setAutoReverse(true);

// Створення повороту об'єкту
RotateTransition rotateTransition =
    new RotateTransition(Duration.millis(3000), rectParallel);
rotateTransition.setByAngle(180f);
rotateTransition.setCycleCount(4);
rotateTransition.setAutoReverse(true);

// Масштабування об'єкту
ScaleTransition scaleTransition =
    new ScaleTransition(Duration.millis(2000), rectParallel);
scaleTransition.setToX(2f);
scaleTransition.setToY(2f);
scaleTransition.setCycleCount(2);
scaleTransition.setAutoReverse(true);

// Створення можливості паралельно виконувати анімацію
ParallelTransition parallelTransition =
    new ParallelTransition();
parallelTransition.getChildren().addAll(
        fadeTransition,
        translateTransition,
        rotateTransition,
        scaleTransition
    );

```

### Реалізація сцени з поєднанням розглянутих технік анімації, обробки зображення та малювання за допомогою примітивів в *JavaFX*

```

package lab3;

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

import javafx.animation.FadeTransition;
import javafx.animation.ParallelTransition;
import javafx.animation.PathTransition;
import javafx.animation.RotateTransition;
import javafx.animation.ScaleTransition;
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.Circle;
import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.scene.shape.Rectangle;
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) // перевизначений стандартний
конструктор
    {
        this.image = image;
    }

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

        ReadingImageFromFile.LoadBitmapImage("D:/Eclipse_workspace/JavaFX_Lab2_for_Com
p_Graphics_Labs/sources/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);
        scene.setFill(Color.BLACK);
        Circle cir;

        int let = 0;
        int let1 = 0;
        int let2 = 0;
        char[][] map = new char[width][height];
        // виконуємо зчитування даних про пікселі
        BufferedInputStream reader = new BufferedInputStream (new
FileInputStream("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
                {
                    cir = new Circle ((j)*2,(height-1-i),1,
Color.valueOf((returnPixelColor(let1)))); // за допомогою стандартного
// примітива Коло радіусом в 1 піксель та кольором
визначеним за допомогою методу returnPixelColor малюємо піксель

```

```

        //root.getChildren().add(cir); //додаємо об'єкт в
сцену
        if (returnPixelColor(let1) == "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
    {
        cir = new Circle ((j)*2+1,(height-1-
i),1,Color.valueOf((returnPixelColor(let2))));
        //root.getChildren().add(cir);
        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); // ініціалізуємо сцену
primaryStage.show(); // візуалізуємо сцену

reader.close();

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

BufferedOutputStream writer = new BufferedOutputStream (new
FileOutputStream("map.txt")); // записуємо карту для руху по траєкторії 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);

```

```

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])
    );
}

final Rectangle rectPath = new Rectangle (0, 0, 60, 60);
rectPath.setArcHeight(10);
rectPath.setArcWidth(10);
rectPath.setFill(Color.ORANGE);
root.getChildren().add(rectPath);

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

Arc arc1 = new Arc(40,340,40,40,30,200);
arc1.setFill(Color.YELLOW);
root.getChildren().add(arc1);

Arc arc2 = new Arc(42,340,40,40,160,170);
arc2.setFill(Color.YELLOW);
root.getChildren().add(arc2);

Circle circ = new Circle (40,340,10, Color.WHITE);
circ.setStroke(Color.BLACK);
circ.setStrokeWidth(2);
root.getChildren().add(circ);

Arc arc3 = new Arc(40,340,10,10,30,150);
arc1.setFill(Color.YELLOW);
root.getChildren().add(arc3);

Circle cir1 = new Circle (40,340,3, Color.BLACK);
root.getChildren().add(cir1);

RotateTransition rotForArc1 =
    new RotateTransition(Duration.millis(500), arc1);
rotForArc1.setByAngle(20f);
rotForArc1.setCycleCount(20);
rotForArc1.setAutoReverse(true);

RotateTransition rotForArc2 =
    new RotateTransition(Duration.millis(500), arc2);
rotForArc2.setByAngle(-20f);
rotForArc2.setCycleCount(20);
rotForArc2.setAutoReverse(true);

TranslateTransition tran = new
TranslateTransition(Duration.seconds(4.0),arc3);
tran.setFromX(0);
tran.setToX(200);
tran.play();

TranslateTransition tran4 = new
TranslateTransition(Duration.seconds(4.0),cir1);

```

```

        tran4.setFromX(0);
        tran4.setToX(200);
        tran4.play();

        TranslateTransition tran1 = new
TranslateTransition(Duration.seconds(4.0), arc1);
        tran1.setFromX(0);
        tran1.setToX(200);
        tran1.play();

        TranslateTransition tran2 = new
TranslateTransition(Duration.seconds(4.0), arc2);
        tran2.setFromX(0);
        tran2.setToX(200);
        tran2.play();

        TranslateTransition tran3 = new
TranslateTransition(Duration.seconds(4.0), circ);
        tran3.setFromX(0);
        tran3.setToX(200);
        tran3.play();

        FadeTransition fadeTransition =
            new FadeTransition(Duration.seconds(12.0), rectPath);
        fadeTransition.setFromValue(1.0f);
        fadeTransition.setToValue(0.0f);
        fadeTransition.setCycleCount(1);

        ScaleTransition scaleTransition =
            new ScaleTransition(Duration.seconds(20.0),
rectPath);

        scaleTransition.setToX(-1f);
        scaleTransition.setToY(-1f);

        ParallelTransition parallelTransition =
            new ParallelTransition();
        parallelTransition.getChildren().addAll(
            tran,
            tran1,
            tran2,
            tran3,
            tran4,
            fadeTransition,
            rotForArc1,
            rotForArc2,
            scaleTransition
        );

        parallelTransition.play();

    }

    private String returnPixelColor (int color) // метод для співставлення
кольорів 16-бітного зображення
    {
        String col = "BLACK";
        switch(color)
        {
            case 0: return "BLACK";    //BLACK;

```

```

        case 1: return "LIGHTCORAL"; //LIGHTCORAL;
        case 2: return "GREEN";      //GREEN
        case 3: return "BROWN";      //BROWN
        case 4: return "BLUE";       //BLUE;
        case 5: return "MAGENTA";    //MAGENTA;
        case 6: return "CYAN";       //CYAN;
        case 7: return "LIGHTGRAY";  //LIGHTGRAY;
        case 8: return "DARKGRAY";   //DARKGRAY;
        case 9: return "RED";         //RED;
        case 10: return "LIGHTGREEN"; //LIGHTGREEN
        case 11: return "YELLOW";     //YELLOW;
        case 12: return "LIGHTBLUE";  //LIGHTBLUE;
        case 13: return "LIGHTPINK";  //LIGHTMAGENTA
        case 14: return "LIGHTCYAN";  //LIGHTCYAN;
        case 15: return "WHITE";      //WHITE;
    }
    return col;
}

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

```

## Завдання

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






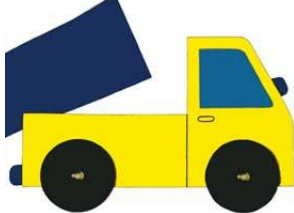
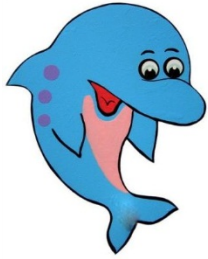




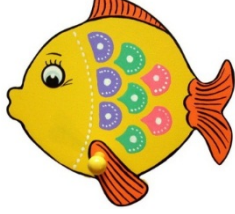


Обов'язковою є реалізація таких видів анімації:

- 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).











## Варіанти завдань

Варіант	Сцена	Варіант	Сцена
1		8	
2		9	
3		10	
4		11	
5		12	
6		13	
7		14	

15		18	
16		19	
17		20	

### Додаткові варіанти завдань

Варіант	Сцена	Варіант	Сцена
1		5	
2		6	
3		7	
4		8	

## Контрольні питання

- 1) Що таке *.bmp*?
- 2) Опишіть структуру файлу *.bmp*.
- 3) З яких частин складається заголовок файлу?
- 4) З яких частин складається заголовок зображення?
- 5) Чим відрізняється *big-endian* та *little-endian*?
- 6) Назвіть переваги та недоліки засобів *JavaFX* для відображення файлів *.bmp*.
- 7) Назвіть методи класу *Image*, *ImageView* та *WritableImage*.
- 8) Назвіть 5 видів анімації в *JavaFX*.