НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ І. СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

ЛАБОРАТОРНА РОБОТА №3

з дисципліни

"МАТЕМАТИЧНІ ТА АЛГОРИТМІЧНІ ОСНОВИ КОМП’ЮТЕРНОЇ ГРАФІКИ"

ТЕМА: «Структура файлів формату .bmp. Анімація примітивів за допомогою засобів бібліотеки JavaFX»

Підготував:

студент групи КП-71

Пилипенко І.О.

Київ-2020

**Код**

package lab3;  
  
import javafx.animation.\*;  
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.\*;  
import javafx.stage.Stage;  
import javafx.util.Duration;  
  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
public class Computer extends Application {  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) throws FileNotFoundException {  
 Group root = new Group();  
 Scene scene = new Scene(root, 1200, 600);  
  
 Polygon p0 = new Polygon(340,230, 435, 230, 387.5, 350);  
 p0.setFill(Color.*BLACK*);  
 root.getChildren().add(p0);  
  
 Polygon p01 = new Polygon(341,231, 434, 231, 387.5, 349);  
 p01.setFill(Color.*WHITE*);  
 root.getChildren().add(p01);  
  
 Rectangle r0 = new Rectangle(340, 292, 95,20);  
 r0.setFill(Color.*BLACK*);  
 root.getChildren().add(r0);  
  
 Rectangle r01 = new Rectangle(341, 293, 93,18);  
 r01.setFill(Color.*WHITE*);  
 root.getChildren().add(r01);  
  
 Polygon p1 = new Polygon(300,180, 445, 160, 460, 270, 315, 290);  
 p1.setFill(Color.*BLACK*);  
 root.getChildren().add(p1);  
  
 Polygon p2 = new Polygon(301,181, 444, 161, 459, 269, 316, 289);  
 p2.setFill(Color.*WHITE*);  
 root.getChildren().add(p2);  
  
 Polygon p3 = new Polygon(315,195, 430, 175, 445, 255, 330, 275);  
 p3.setFill(Color.*CYAN*);  
 root.getChildren().add(p3);  
  
 Rectangle r1 = new Rectangle(300, 300, 175,52);  
 r1.setFill(Color.*BLACK*);  
 root.getChildren().add(r1);  
  
 Rectangle r2 = new Rectangle(301, 301, 173,50);  
 r2.setFill(Color.*CYAN*);  
 root.getChildren().add(r2);  
  
 Rectangle r11 = new Rectangle(315, 308, 25,4);  
 r11.setFill(Color.*WHITE*);  
 root.getChildren().add(r11);  
  
 Rectangle r12 = new Rectangle(400, 312, 25,4);  
 r12.setFill(Color.*WHITE*);  
 root.getChildren().add(r12);  
 Rectangle r13 = new Rectangle(430, 312, 25,4);  
 r13.setFill(Color.*WHITE*);  
 root.getChildren().add(r13);  
  
 final ImageView imv = new ImageView();  
 final Image image2 = new Image(new FileInputStream("face.png"));  
 imv.setImage(image2);  
 imv.setX(220);  
 imv.setY(130);  
 double size = 220;  
 imv.setFitHeight(size);  
 imv.setFitWidth(size\*1.5);  
 root.getChildren().add(imv);  
  
 // Animation  
 int cycleCount = 2;  
 int time = 3000;  
  
 ScaleTransition scaleTransition = new ScaleTransition(Duration.*millis*(time), root);  
 scaleTransition.setToX(2);  
 scaleTransition.setToY(2);  
 scaleTransition.setAutoReverse(true);  
  
 RotateTransition rotateTransition = new RotateTransition(Duration.*millis*(time), root);  
 rotateTransition.setByAngle(360f);  
 rotateTransition.setCycleCount(cycleCount);  
 rotateTransition.setAutoReverse(true);  
  
 TranslateTransition translateTransition = new TranslateTransition(Duration.*millis*(time), root);  
 translateTransition.setFromX(150);  
 translateTransition.setToX(500);  
 translateTransition.setCycleCount(cycleCount + 1);  
 translateTransition.setAutoReverse(true);  
  
 TranslateTransition translateTransition2 = new TranslateTransition(Duration.*millis*(time), root);  
 translateTransition2.setFromX(50);  
 translateTransition2.setToX(500);  
 translateTransition2.setCycleCount(cycleCount + 1);  
 translateTransition2.setAutoReverse(true);  
  
 ScaleTransition scaleTransition2 = new ScaleTransition(Duration.*millis*(time), root);  
 scaleTransition2.setToX(0.1);  
 scaleTransition2.setToY(0.1);  
 scaleTransition2.setCycleCount(cycleCount);  
 scaleTransition2.setAutoReverse(true);  
  
 ParallelTransition parallelTransition = new ParallelTransition();  
 parallelTransition.getChildren().addAll(  
 rotateTransition,  
 scaleTransition,  
 scaleTransition2,  
 translateTransition  
 );  
 parallelTransition.setCycleCount(Timeline.*INDEFINITE*);  
 parallelTransition.play();  
  
  
 primaryStage.setResizable(false);  
 primaryStage.setTitle("Lab 3");  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

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 setSize (long size)  
 {  
 this.size = size;  
 }  
  
 public long getSize ()  
 {  
 return size;  
 }  
  
 public void setReserveField1 (short reserveField1)  
 {  
 this.reserveField1 = reserveField1;  
 }  
  
 public short getReserveField1 ()  
 {  
 return reserveField1;  
 }  
  
 public void setReserveField2 (short reserveField2)  
 {  
 this.reserveField2 = reserveField2;  
 }  
  
 public short getReserveField2 ()  
 {  
 return reserveField2;  
 }  
  
 public void setOffset (long offset)  
 {  
 this.offset = offset;  
 }  
  
 public long getOffset ()  
 {  
 return offset;  
 }  
  
 public void setSizeOfHeader (long sizeOfHeader)  
 {  
 this.sizeOfHeader = sizeOfHeader;  
 }  
  
 public long getSizeOfHeader ()  
 {  
 return sizeOfHeader;  
 }  
  
 public void setWidth (long width)  
 {  
 this.width = width;  
 }  
  
 public long getWidth ()  
 {  
 return width;  
 }  
  
 public void setHeight (long height)  
 {  
 this.height = height;  
 }  
  
 public long getHeight ()  
 {  
 return height;  
 }  
  
 public void setNumberOfColorPlanes (short numberOfColorPlanes)  
 {  
 this.numberOfColorPlanes = numberOfColorPlanes;  
 }  
  
 public short getNumberOfColorPlanes ()  
 {  
 return numberOfColorPlanes;  
 }  
  
 public void setBitsCount (short bitsCount)  
 {  
 this.bitsCount = bitsCount;  
 }  
  
 public short getBitsCount ()  
 {  
 return bitsCount;  
 }  
  
 public void setCompression (long compression)  
 {  
 this.compression = compression;  
 }  
  
 public long getCompression ()  
 {  
 return compression;  
 }  
  
 public void setSizeOfCompImage (long sizeOfCompImage)  
 {  
 this.sizeOfCompImage = sizeOfCompImage;  
 }  
  
 public long getSizeOfCompImage ()  
 {  
 return sizeOfCompImage;  
 }  
  
 public void setHorizontalResolution (long horizontalResolution)  
 {  
 this.horizontalResolution = horizontalResolution;  
 }  
  
 public long getHorizontalResolution ()  
 {  
 return horizontalResolution;  
 }  
  
 public void setVerticalResolution (long verticalResolution)  
 {  
 this.verticalResolution = verticalResolution;  
 }  
  
 public long getVerticalResolution ()  
 {  
 return verticalResolution;  
 }  
  
 public void setNumbOfUsedColors (long numbOfUsedColors)  
 {  
 this.numbOfUsedColors = numbOfUsedColors;  
 }  
  
 public long getNumbOfUsedColors ()  
 {  
 return numbOfUsedColors;  
 }  
  
 public void setNumbOfImportantColors (long numbOfImportantColors)  
 {  
 this.numbOfImportantColors = numbOfImportantColors;  
 }  
  
 public long getNumbOfImportantColors ()  
 {  
 return numbOfImportantColors;  
 }  
  
 public void setHalfOfWidth (long halfOfWidth)  
 {  
 this.halfOfWidth = halfOfWidth;  
 }  
  
 public long getHalfOfWidth ()  
 {  
 return 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 )  
 // 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);  
 setNumbOfUsedColors(nUsCol);  
 setNumbOfImportantColors(nImpCol);  
 setHalfOfWidth(half);  
 }  
}

package sample;  
  
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{  
 public HeaderBitmapImage image;  
 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*("par.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();  
  
 // writing  
 BufferedOutputStream writer = new BufferedOutputStream (new FileOutputStream("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  
 {  
 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  
 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);  
 }  
}

package lab3;  
  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.layout.StackPane;  
import javafx.stage.Stage;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.image.PixelReader;  
import javafx.scene.paint.Color;  
  
public class ReadingFromFileBMPWithStandart extends Application {  
   
 @Override  
 public void start(Stage primaryStage) {  
   
 // Create Image and ImageView objects  
 Image image = new Image("par.bmp");  
 ImageView imageView = new ImageView();  
 imageView.setImage(image);  
   
 // Obtain PixelReader  
 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());  
   
 // 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());  
 }  
 }  
   
 // Display image on screen  
 StackPane root = new StackPane();  
 root.getChildren().add(imageView);  
 Scene scene = new Scene(root, image.getWidth(), image.getHeight());  
 primaryStage.setTitle("Image Read Test");  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
   
 public static void main(String[] args) {  
 *launch*(args);  
 }  
}

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(){ }  
  
 // 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 = 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);  
 // 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("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 = 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 переводу 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;  
 }  
}

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 пікселі 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.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 { }  
}

package lab3;  
  
import javafx.animation.FadeTransition;  
import javafx.animation.Interpolator;  
import javafx.animation.KeyFrame;  
import javafx.animation.KeyValue;  
import javafx.animation.ParallelTransition;  
import javafx.animation.PathTransition;  
import javafx.animation.RotateTransition;  
import javafx.animation.ScaleTransition;  
import javafx.animation.SequentialTransition;  
import javafx.animation.Timeline;  
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.CubicCurveTo;  
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 Transitions extends Application {  
  
 public static void main (String args[]) {  
 *launch*(args);  
 }  
  
 public void start(Stage primaryStage) throws Exception {  
 Group root = new Group();  
 Scene scene = new Scene (root, 500, 500);  
  
 //Створення прямокутника червоного кольору  
 final Rectangle rect1 = new Rectangle(10, 10, 100, 100);  
 rect1.setArcHeight(20);  
 rect1.setArcWidth(20);  
 rect1.setFill(Color.*RED*);  
 root.getChildren().add(rect1);  
  
 // створення ефекту зникнення  
 FadeTransition ft = new FadeTransition(Duration.*millis*(3000), rect1);  
 ft.setFromValue(1.0); // встановлення початкового значення прозорості об'єкту  
 ft.setToValue(0.1); // встановлення кінцевого значення прозорості об'єкту  
 ft.setCycleCount(Timeline.*INDEFINITE*);  
 ft.setAutoReverse(true);  
 ft.play();  
  
 // 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 траекторії з 2 ліній типу CubicCurveTo  
 Path path = new Path();  
 path.getElements().add(new MoveTo(50,40)); // вказання початкової позиції, з якої починається траекторія  
 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 траекторії  
 PathTransition pathTransition = new PathTransition();  
 pathTransition.setDuration(Duration.*millis*(4000)); // встановлення часу анмації  
 pathTransition.setPath(path); // прив'язування траекторії  
 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  
  
 //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*(3000), rectParallel);  
 translateTransition.setFromX(40);  
 translateTransition.setToX(360);  
 translateTransition.setCycleCount(3);  
 translateTransition.setAutoReverse(true);  
  
 // Створення повороту об'єкту  
 RotateTransition rotateTransition =  
 new RotateTransition(Duration.*millis*(3000), rectParallel);  
 rotateTransition.setByAngle(180f);  
 rotateTransition.setCycleCount(3);  
 rotateTransition.setAutoReverse(true);  
  
 // Масштабування об'єкту  
 ScaleTransition scaleTransition =  
 new ScaleTransition(Duration.*millis*(3000), 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  
 );  
  
 parallelTransition.setCycleCount(Timeline.*INDEFINITE*);  
 parallelTransition.play();  
  
 Rectangle rectSeq = new Rectangle(25,25,50,50);  
 rectSeq.setArcHeight(15);  
 rectSeq.setArcWidth(15);  
 rectSeq.setFill(Color.*CRIMSON*);  
 rectSeq.setTranslateX(50);  
 rectSeq.setTranslateY(50);  
 root.getChildren().add(rectSeq);  
  
 fadeTransition =  
 new FadeTransition(Duration.*millis*(1000), rectSeq);  
 fadeTransition.setFromValue(1.0f);  
 fadeTransition.setToValue(0.3f);  
 fadeTransition.setCycleCount(1);  
 fadeTransition.setAutoReverse(true);  
  
 translateTransition =  
 new TranslateTransition(Duration.*millis*(2000), rectSeq);  
 translateTransition.setFromX(50);  
 translateTransition.setFromY(40);  
 translateTransition.setToX(375);  
 translateTransition.setToY(375);  
 translateTransition.setCycleCount(1);  
 translateTransition.setAutoReverse(true);  
  
 rotateTransition =  
 new RotateTransition(Duration.*millis*(2000), rectSeq);  
 rotateTransition.setByAngle(180f);  
 rotateTransition.setCycleCount(4);  
 rotateTransition.setAutoReverse(true);  
   
 /\*  
 scaleTransition =   
 new ScaleTransition(Duration.millis(2000), rectSeq);  
 scaleTransition.setFromX(1);  
 scaleTransition.setFromY(1);  
 scaleTransition.setToX(2);  
 scaleTransition.setToY(2);  
 scaleTransition.setCycleCount(1);  
 scaleTransition.setAutoReverse(true);  
 \*/  
  
 SequentialTransition sequentialTransition = new SequentialTransition();  
 sequentialTransition.getChildren().addAll(  
 fadeTransition,  
 translateTransition  
 );  
  
 sequentialTransition.setCycleCount(Timeline.*INDEFINITE*);  
 sequentialTransition.setAutoReverse(true);  
 sequentialTransition.play();  
  
 final Rectangle rectBasicTimeline = new Rectangle(100, 60, 150, 50);  
 rectBasicTimeline.setFill(Color.*RED*);  
 root.getChildren().add(rectBasicTimeline);  
  
 final Timeline timeline = new Timeline();  
 timeline.setCycleCount(Timeline.*INDEFINITE*);  
 timeline.setAutoReverse(true);  
 final KeyValue kv = new KeyValue(rectBasicTimeline.xProperty(), 300);  
 final KeyFrame kf = new KeyFrame(Duration.*millis*(1000), kv);  
 timeline.getKeyFrames().add(kf);  
 timeline.play();  
  
 final Rectangle rectBasicTimeline1 = new Rectangle(200, 100, 200, 100);  
 rectBasicTimeline1.setFill(Color.*BROWN*);  
 root.getChildren().add(rectBasicTimeline1);  
 final Timeline timeline1 = new Timeline();  
 timeline1.setCycleCount(Timeline.*INDEFINITE*);  
 timeline1.setAutoReverse(true);  
 final KeyValue kv1 = new KeyValue(rectBasicTimeline1.xProperty(), 400,  
 Interpolator.*EASE\_BOTH*);  
 final KeyFrame kf1 = new KeyFrame(Duration.*millis*(700), kv1);  
 timeline1.getKeyFrames().add(kf1);  
 timeline1.play();  
  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

**Малюнки та анімації**

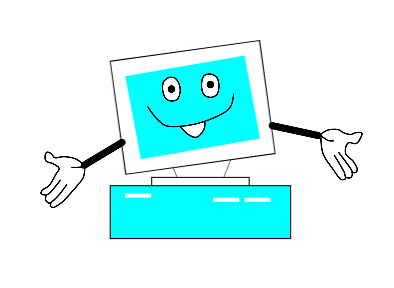
****

Рис1. Малюнок

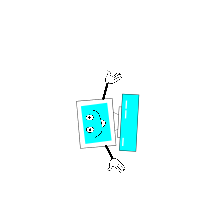
****

Рис2. Анімація