CAPITOLUL 6

6.1. Crearea interfețelor grafice.

În Java există 3 pachete de bază pentru crearea interfețelor grafice (Graphics User Interface). Acestea sunt 1-AWT (Abstract Windows Toolkit) 2-Swing și JavaFX

Swing — este o bibliotecă destinată creării interfețelor grafice pentru programele în limbajul Java. Reprezintă o tehnologie alternativă classelor – AWT a primit denumirea de tehnologia SWING, conține o serie de classe (Swing API), a fost creată în bază bibliotecii AWT cu scopul înlăturării neajunsurilor acesteia, de exemplu: - numărul redus a componentelor grafice; - dependența copmportării și reprezentării interfeței grafice AWT de Sistemul de Operare.

Tehnologia Swing pune la dispoziție mecanizmele de dirijare cu următoarele aspecte a prezentării:

- Tastatura (Swing presupune modalitatea captării introducerii de la tastatură)
- Culoarea (Swing presupune modalitatea schimbării culorii pe care o vedem la ecran)
- Text Boxul (Swing pune la dispoziție componentele text pentru rezolvarea problemelor zilnice)

JComponent

Clasa de bază a tuturor componentelor vizuale Swing este clasa JComponent, care este o super-clasa și este abstract, deci Swing-ul poate folosi toate componentele acestuia. JComponent deasemenea conține metoda add(), care ne permite adăugarea altor obiecte a clasei JComponent, în așa fel putem adăuga orice component Swing la oricare altul pentru crearea componentelor incorporate (de exemplu JPanel conține JButton).

Componentele folosite pentru crearea interfețelor grafice Swing pot fi grupate astfel:

• Componente atomice: JLabel, JButton, JCheckBox, JRadioButton, JToggleButton, JScrollBar, JSlider, JProgressBar, JSeparator

• Componente complexe: JTable, JTree, JComboBox, JSpinner, JList, JFileChooser, JColorChooser, JOptionPane

• Componente pentru editare de text: JTextField, JFormattedTextField, JPasswordField, JTextArea, JEditorPane, JTextPane

• Meniuri: JMenuBar, JMenu, JPopupMenu, JMenuItem, JCheckboxMenuItem, JRadioButtonMenuItem

• Containere intermediare: JPanel, JScrollPane, JSplitPane, JTabbedPane, JDesktopPane, JToolBar

• Containere de nivel înalt: JFrame, JDialog, JWindow, JInternalFrame, JApplet

Metoda Paint

Toate desenele care trebuie să apară pe o suprafață de afișare se realizează în metoda Paint a unei componente. Metoda Paint este definită în superclasa Component însă nu are nici o implementare și, din acest motiv, orice obiect grafic care dorește să se deseneze trebuie să o supradefinească pentru a-și crea propria sa reprezentare. Componentele standard AWT au deja supradefinită această metodă deci nu trebuie să ne preocupe desenarea lor, însă putem modifica reprezentarea lor grafică prin crearea unei subclase și supradefinirea metodei Paint, având însă grijă să apelăm și metoda superclasei care se ocupă cu desenarea efectivă a componentei. În exemplul de mai jos, redefinim metoda paint pentru un obiect de tip Frame, pentru a crea o clasă ce instanțiază ferestre pentru o aplicație demonstrativă (în colțul stânga sus este afișat textul ”Aplicatie DEMO”).

```
import java . awt .*;

class Fereastra extends Frame {

public Fereastra ( String titlu ) {

super ( titlu );

setSize (200 , 100) ;

}

public void paint ( Graphics g) {

// Apelam metoda paint a clasei Frame

super . paint (g);

g. setFont (new Font (" Arial ", Font .BOLD , 11));

g. setColor ( Color .red );

g. drawString (" Aplicatie DEMO ", 5, 35);
```

```
}

}

public class TestPaint {

public static void main ( String args []) {

Fereastra f = new Fereastra (" Test paint ");

f. show ();

}

}
```

Suprafețe de desenare - clasa Canvas

În afara posibilității de a utiliza componente grafice standard, Java oferă și posibilitatea controlului la nivel de punct (pixel) pe dispozitivul grafic, respectiv desenarea a diferitor forme grafice direct pe suprafața unei componente. Deși este posibil, în general nu se desenează la nivel de pixel direct pe suprafața ferestrelor sau a altor containere, ci vor fi folosite clase dedicate acestui scop.

În AWT a fost definit un tip special de componentă numită Canvas (pânză de pictor), al cărei scop este de a fi extinsă pentru a implementa obiecte grafice cu o anumită înfățișare. Așadar, Canvas este o clasă generică din care se derivează subclasele pentru crearea suprafețelor de desenare (planșe). Planșele nu pot conține alte componente grafice, ele fiind utilizate doar ca suprafețe de desenat sau ca fundal pentru animație. Desenarea pe o planșa se face prin supradefinirea metodei paint a acesteia. Concret, o planșă este o suprafață dreptunghiulară de culoare albă, pe care se poate desena. Dimensiunile sale implicite sunt 0 și, din acest motiv, este recomandat ca o planșa să redefinească metoda getPreferredSize, eventual și getMinimumSize, getMaximumSize, deoarece acestea vor fi apelate de către gestionarii de poziționare.

Etapele uzuale care trebuie parcurse pentru crearea unui desen, sau mai bine zis a unei componente cu o anumită înfățișare, sunt:

• crearea unei planșe de desenare, adică o subclasă a lui Canvas;

• redefinirea metodei paint din clasa respectivă;

• redefinirea metodelor getPreferredSize, eventual getMinimumSize, getMaximumSize;

• adăugarea planșei pe un container cu metoda add.

• tratarea evenimentelor de tip FocusEvent, KeyEvent, MouseEvent, ComponentEvent, dacă este cazul.

Definirea generică a unei planșe are următorul format:

Să definim o planșă pe care desenăm un pătrat și cercul său circumscris, colorate diferite. La fiecare click de mouse, vom interschimba cele două culori între ele.

Listing 10.2: Folosirea clasei Canvas

```
import java . awt .*;

import java . awt. event .*;

class Plansa extends Canvas {

Dimension dim = new Dimension (100 , 100) ;

private Color color [] = { Color .red , Color . blue };

private int index = 0;

public Plansa () {

this . addMouseListener (new MouseAdapter () {

public void mouseClicked ( MouseEvent e) {

index = 1 - index ;

repaint ();

}

});

}

public void paint ( Graphics g) {

g. setColor ( color [ index ]);

g. drawRect (0, 0, dim .width , dim. height );
```

```
g. setColor ( color [1 - index ]);

g. fillOval (0, 0, dim .width , dim. height );

}

public Dimension getPreferredSize () {

return dim ;

}

}

class Fereastra extends Frame {

public Fereastra ( String titlu ) {

super ( titlu );

setSize (200 , 200) ;

add (new Plansa () , BorderLayout . CENTER );

}

}

public class TestCanvas {

public static void main ( String args []) {

new Fereastra (" Test Canvas "). show ();

}

}
```

Proprietățile contextului grafic

La orice tip de desenare parametrii legați de culoare, font, etc. vor fi specificați pentru contextul grafic în care se face desenarea și nu vor fi trimiși ca argumentele metodelor respective de desenare. In continuare, enumerăm aceste proprietăți și metodele asociate lor din clasa Graphics.

Proprietate Metode

Culoarea de desenare Color getColor() void setColor(Color c)

Fontul de scriere a textelor Font getFont() void setFont(Font f)

Originea coordonatelor translate(int x, int y) Zona de decupare Shape getClip() (zona în care sunt vizibile desenele) void setClip(Shape s)

Modul de desenare void setXorMode(Color c) void setPaintMode(Color c)

Primitive grafice

Prin primitive grafice ne vom referiîn continuare la metodele clasei Graphics, care permit desenarea figurilor geometrice și texte. Desenarea textelor se face uzual cu metoda drawString care primește ca argumente un șir și colțul din stânga-jos al textului. Textul va fi desenat cu fontul și culoarea curente ale contextului grafic.

// Desenam la coordonatele x=10, y=20;

drawString("Hello", 10, 20);

Desenarea figurilor geometrice se realizează cu următoarele metode:

Figură geometrică Metode

Linie drawLine

drawPolyline

Dreptunghi simplu drawRect

fillRect

clearRect

Dreptunghi cu chenar draw3DRect

”ridicat” sau ”adâncit” fill3DRect

Dreptunghi cu colțuri drawRoundRect

Retunjite fillRoundRect

Poligon drawPolygon

fillPolygon

Oval (Elipsă) drawOval

fillOval

Arc circular sau drawArc

Eliptic fillArc

Clasa Font

Un obiect de tip Font încapsulează informații despre toți parametrii unui font, mai puțin despre metrica acestuia. Constructorul uzual al clasei este cel care primește ca argument numele fontului, dimensiunea și stilul acestuia:

Font(String name, int style, int size)

Stilul unui font este specificat prin intermediul constantelor: Font.PLAIN, Font.BOLD, Font.ITALIC iar dimensiunea printr-un întreg, ca în exemplele de mai jos:

```
new Font("Dialog", Font.PLAIN, 12);

new Font("Arial", Font.ITALIC, 14);

new Font("Courier", Font.BOLD, 10);
```

Folosirea unui obiect de tip Font se realizează uzual astfel:

```
// Pentru componente etichetate

Label label = new Label("Un text");

label.setFont(new Font("Dialog", Font.PLAIN, 12));

// In metoda paint(Graphics g)

g.setFont(new Font("Courier", Font.BOLD, 10));

g.drawString("Alt text", 10, 20);
```

O platformă de lucru are instalate, la un moment dat, o serie întreagă de fonturi care sunt disponibile pentru scrierea textelor. Lista acestor fonturi se poate obține astfel:

Font[] fonturi = GraphicsEnvironment.

getLocalGraphicsEnvironment().getAllFonts();

Exemplul urmator afișează lista tuturor fonturilor disponibile pe platform curentă de lucru. Textul fiecărui nume de font va fi scris cu fontul său corespunzător.

```
import java . awt .*;

class Fonturi extends Canvas {

private Font [] fonturi ;

Dimension canvasSize = new Dimension (400 , 400) ;

public Fonturi () {

fonturi = GraphicsEnvironment .

getLocalGraphicsEnvironment (). getAllFonts ();

canvasSize . height = (1 + fonturi . length ) * 20;

}

public void paint ( Graphics g) {

String nume ;

for (int i=0; i < fonturi . length ; i ++) {

nume = fonturi [i]. getFontName ();

g. setFont (new Font (nume , Font .PLAIN , 14));

g. drawString (i + ". " + nume , 20, (i + 1) * 20);

}

}

public Dimension getPreferredSize () {

return canvasSize ;

}

}
```

```
class Fereastra extends Frame {

public Fereastra ( String titlu ) {

super ( titlu );

ScrollPane sp = new ScrollPane ();

sp. setSize (400 , 400) ;

sp. add( new Fonturi ());

add (sp , BorderLayout . CENTER );

pack ();

}

}

public class TestAllFonts {

public static void main ( String args []) {

new Fereastra ("All fonts "). show ();}}
```

Folosirea ferestrelor

Pentru a fi afișate pe ecran componentele grafice ale unei aplicații trebuie plasate pe o suprafață de afișare (container). Fiecare componentă poate fi conținută doar într-un singur container, adăugarea ei pe o suprafță nouă de afișare determinând eliminarea ei de pe vechiul container pe care fusese plasată. Intrucât containerele pot fi încapsulate în alte containere, o componentă va face parte la un moment dat dintr-o ierarhie. Rădăcina acestei ierarhii trebuie să fie un așa numit container de nivel înalt, care este reprezentat de una din clasele JFrame, JDialog sau JApplet. Intrucât de appleturi ne vom ocupa separat, vom analiza în continuare primele două clase. In general orice aplicație Java independentă bazată pe Swing conține cel puțin un container de nivel înalt reprezentat de fereastra principală a programului, instanță a clasei JFrame. Simplificat, un obiect care reprezintă o fereastră Swing conține o zonă care este rezervată barei de meniuri și care este situată de obieci în partea sa superioară și corpul ferestrei pe care vor fi plasate componentele. Imaginea de mai jos pune în evidență această separare, valabilă de altfel pentru orice container de nivel înalt:

Corpul ferestrei este o instanță a clasei Container ce poate fi obținută cu metoda getContentPane. Plasarea și aranjarea componentelor pe suprafața ferestrei se va face

deci folosind obiectul de tip Container și nu direct fereastra. Așadar, deși este derivată din Frame, clasa JFrame este folosită într-un mod diferit față de părintele său:

```
Frame f = new Frame();

f.setLayout(new FlowLayout());

f.add(new Button("OK"));

JFrame jf = new JFrame();

jf.getContentPane().setLayout(new FlowLayout());

jf.getContentPane().add(new JButton("OK"));
```

Spre deosebire de Frame, un obiect JFrame are un comportament implicit la închiderea ferestrei care constă în ascunderea ferestrei atunci când utilizatorul apasă butonul de închidere. Acest comportament poate fi modificat prin apelarea metodei setDefaultCloseOperation care primește ca argument diverse constante ce se găsesc fie în clasa WindowConstants, fie chiar în JFrame.

```
jf.setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);

jf.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);

jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Adăugarea unei bare de meniuri se realizează cu metoda setJMenuBar, care primește o instanță de tip JMenuBar. Crearea meniurilor este similar cu modelul AWT.

FOLOSIREA MODELELOR

• Prezentarea - modul de reprezentare vizuală a datelor.

• Controlul - transformarea acțiunilor utilizatorului asupra componentelor vizuale în evenimente care să actualizeze automat modelul acestora (datele).

Din motive practice, în Swing părțile de prezentare și control au fost cuplate deoarece exista o legătură prea strânsă între ele pentru a fi concepute ca entități separate. Așadar, arhitectura Swing este de fapt o arhitectură cu model separabil, în care datele componentelor (modelul) sunt separate de reprezentarea lor vizuală. Această

abordare este logică și din perspective faptului că, în general, modul de concepere a unei aplicații trebuie să fie orientat asupra reprezentării și manipulării informațiilor și nu asupra interfeței grafice cu utilizatorul. Pentru a realiza separarea modelului de prezentare, fiecărui obiect corespunz ător unei clase ce descrie o componentă Swing îi este asociat un obiect care gestionează datele sale și care implementează o interfață care reprezintă modelul componentei respective. După cum se observă din tabelul de mai jos, componente cu reprezentări diferite pot avea același tip de model, dar există și componente care au asociate mai multe modele: Model Componentă ButtonModel JButton, JToggleButton, JCheckBox, JRadioButton, JMenu, JMenuItem, JCheckBoxMenuItem, JRadioButtomMenuItem JComboBox ComboBoxModel BoundedRangeModel JProgressBar, JScrollBarm, JSlider JTabbedPane SingleSelectionModel ListModel JList ListSelectionModel JList JTable TableModel JTable TableColumnModel JTree TreeModel JTree TreeSelectionModel Document JEditorPane, JTextPane, JTextArea, JTextField, JPasswordField Folosirea mai multor modele pentru o componenta

```
import javax . swing .*;

import javax . swing . border .*;

import java . awt .*;

import java . awt. event .*;

class Fereastra extends JFrame implements ActionListener {

String data1 [] = {" rosu ", " galben ", " albastru "};

String data2 [] = {"red", " yellow ", " blue "};

int tipModel = 1;

JList lst;

ListModel model1 , model2 ;

public Fereastra ( String titlu ) {

super ( titlu );

setDefaultCloseOperation ( JFrame . EXIT_ON_CLOSE );

// Lista initiala nu are nici un model

lst = new JList ();

getContentPane ().add(lst , BorderLayout . CENTER );
```

```
// La apasara butonului schimbam modelul
JButton btn = new JButton (" Schimba modelul ");
getContentPane ().add(btn , BorderLayout . SOUTH );
btn . addActionListener ( this );
// Cream obiectele corespunzatoare celor doua modele
model1 = new Model1 ();
model2 = new Model2 ();
lst . setModel ( model1 );
pack ();
}
public void actionPerformed ( ActionEvent e) {
if ( tipModel == 1) {
lst . setModel ( model2 );
tipModel = 2;
}
else {
lst . setModel ( model1 );
tipModel = 1;
}
}
// Clasele corespunzatoare celor doua modele
class Model1 extends AbstractListModel {
public int getSize () {
return data1 . length ;
}
public Object getElementAt ( int index ) {
```

```
return data1 [ index ];

}

}

class Model2 extends AbstractListModel {

public int getSize () {

return data2 . length ;

}

public Object getElementAt ( int index ) {

return data2 [ index ];}} }

public class TestModel {

public static void main ( String args []) {

new Fereastra (" Test Model "). show ();

}

}
```

Tratarea evenimentelor

Modelele componentelor trebuie să notifice apariția unor schimbări ale datelor gestionate astfel încât să poată fi reactualizată prezentarea lor sau să fie executat un anumit cod în cadrul unui obiect de tip listener. În Swing, această notificare este realizată în două moduri:

1. Informativ (lightweight) - Modelele trimit un eveniment prin care sunt informați ascultătorii că a survenit o anumită schimbare a datelor, fără a include în eveniment detalii legate de schimbarea survenită. Obiectele de tip listener vor trebui să apeleze metode specifice componentelor pentru a afla ce anume s-a schimbat. Acest lucru se realizează prin intefața ChangeListener iar evenimentele sunt de tip ChangeEvent, modelele care suportă această abordare fiind BoundedRangeModel, ButtonModel și SingleSelectionModel.

Model Listener Tip Eveniment

BoundedRangeModel ChangeListener ChangeEvent

ButtonModel ChangeListener ChangeEvent

SingleSelectionModelModel ChangeListener ChangeEvent

Exemplu

Creați un treiunghi care se rotește în jurul centrului său

Listingul programului:

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
public class Main {
public static void main(String[] args) {
JFrame fr=new JFrame("rotirea treiunghiului în jurul centrului sau ");
fr.setPreferredSize( new Dimension(300,300));
final JPanel pan= new JPanel();
fr.add(pan);
fr.setVisible(true);
fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
fr.pack();
Timer tm= new Timer(500, new ActionListener(){
int i=0;
@Override
public void actionPerformed(ActionEvent arg0) {
Graphics2D gr=(Graphics2D)pan.getRootPane().getGraphics();
pan.update(gr);
GeneralPath path=new GeneralPath();
path.append(new Polygon(new int []{60,-80,50},new int[]{-60,-50,40},3),true);
int x=(60-80+50)/3,y=(-60-50+40)/3;
gr.translate(150, 150);
AffineTransform tranforms = AffineTransform.getRotateInstance((i++)*0.07, x, y);
gr.transform(tranforms);
gr.draw(path);
}});
tm.start();
}
}
```

O aplicație JavaFX este o aplicație Java de bază, ce permite *feature*-uri JavaFX. Codul minim necesar pentru a rula o aplicație JavaFX constă din:

- O clasă ce extinde clasa abstractă `javafx.application.Application`.
- O metodă `main()` ce apelează metoda `launch()` și suprascrie metoda abstractă `start()`. Ca bună practică apelul metodei `launch()` este singurul apel din `main()`
- Un *stage* primar ce este vizibil, ca argument al metodei `start()`

Avem trei tipuri de aplicații JavaFX:

- Aplicații propriu zise, ce folosesc sintaxa Java tradițională și API-ul JavaFX.
- Aplicații FXML. FXML se bazează pe XML și este folosit pentru a defini interfețe-utilizator în aplicații JavaFX. Cu FXML vom defini *layout*-uri statice precum formulare, controale sau tabele. Putem construi, de asemenea, *layout*-uri dinamice prin includerea unui *script*.
- Aplicații *preloader*, folosite în procesul de *deployment*.

Un stage (`javafx.stage.Stage`) este un *container* GUI *top level* pentru toate obiectele grafice, iar un scene (`javafx.scene.Scene`) este *container*-ul de bază. *Stage*-ul primar este construit de platformă, dar pot fi construite și alte obiecte *stage* de către aplicație. Obiectele *stage* trebuie să fie construite și modificate în firul aplicației JavaFX.

Articolele individuale care se află în interiorul scenei grafice sunt numite noduri. Fiecare nod este clasificat ca fiind:

- Branch sau un părinte, ceea ce înseamnă că poate avea descendenți.
- O frunză.

Primul nod din arbore este numit rădacină și nu are părinte.

Scena grafică este așadar o structură arborescentă. API-ul JavaFX face ca interfața grafică să fie mai ușor de creat, mai ales când sunt implicate efecte vizuale complexe și transformări.

API-ul scenei grafice JavaFX este un *retained mode*, adică el gestionează un model intern al tuturor obiectelor grafice din aplicație. În orice moment el știe ce obiecte să afișeze, ce zone ale ecranului necesită redesenare și cum să fie *renderizat* în cel mai eficient mod. Aceasta reduce semnificativ cantitatea de cod necesar în aplicație.

În aplicația dată adăugarea unui buton cu text și al unui *container* de layout. Butonul va fi un nod frunză, iar StackPane-ul nod rădăcină.

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class JavaFXApplication extends Application {
    public static void main(String[] args) {
        launch(args);
```

```
}

@Override
public void start(Stage primaryStage) {

primaryStage.setTitle("Hello World!");
    Button btn = new Button();
    btn.setText("Say 'Hello World'");
    btn.setOnAction(new EventHandler() {

  @Override
  public void handle(ActionEvent event) {
      System.out.println("Hello World!");
        }
    });

   StackPane root = new StackPane();
   root.getChildren().add(btn);
   primaryStage.setScene(new Scene(root, 300, 250));
   primaryStage.show();
}
```

Interfața JavaFX

O aplicatie JavaFX este o instanta a clasei Application
public abstract class Application extends Object;
▪ Instantierea unui obiectApplication se face prin executarea metodei statice launch()
public static void launch(String... args);
args parametrii aplicatiei(parametrii metodei main).
▪ JavaFX runtime execută urmatoarele operatiuni:
1. Creazaun obiectApplication
2. Apeleaza metoda init a obiectului Application
3. Apeleaza metoda start a obiectului Application
4. Asteapta sfarsitul aplicatiei
▪ Parametrii aplicatiei sunt obtinuti prin metoda getParameters()

Exemplu:

```
public class Main extends Application {
@Override public void start(Stage stage)
 { Group root = new Group();
 Scene scene = new Scene(root, 500, 500, Color.PINK);
 stage.setTitle("Welcome to JavaFX!");
stage.setScene(scene); stage.show();
}
public static void main(String[] args)
 { launch(args);
//se creaza un obiect de tip Application
 }
}
```

Adăugarea nodurilor

```
// Cream un nod de tip Group
 Group group = new Group();
// Cream un nod de tip Rectangle
 Rectangle r = new Rectangle(25,25,50,50);
 r.setFill(Color.BLUE); group.getChildren().add(r);
// Cream un nod de tip Circle
 Circle c = new Circle(200,200,50, Color.web("blue", 0.5f));
 group.getChildren().add(c);
```

Gestionarea poziționării componentelor UI HBOX pe linii orizontală

```
HBox root = new HBox(5);
 root.setPadding(new Insets(100));
 root.setAlignment(Pos.BASELINE_RIGHT);
Button prevBtn = new Button("Previous");
 Button nextBtn = new Button("Next");
Button cancBtn = new Button("Cancel");
 Button helpBtn = new Button("Help");
root.getChildren().addAll(prevBtn, nextBtn, cancBtn, helpBtn);
```

Gestionarea poziționării componentelor UI ▪ VBOX pe coloane vertical

```
VBox root = new VBox(5);
 root.setPadding(new Insets(20));
 root.setAlignment(Pos.BASELINE_LEFT);
Button prevBtn = new Button("Previous");
 Button nextBtn = new Button("Next");
Button cancBtn = new Button("Cancel");
 Button helpBtn = new Button("Help");
root.getChildren().addAll(prevBtn, nextBtn, cancBtn, helpBtn);
Scene scene = new Scene(root, 150, 200);
```

Gestionarea poziționării componentelor UI ▪ AnchorPane pe poziții

```
AnchorPane root = new AnchorPane();
Button okBtn = new Button("OK");
 Button closeBtn = new Button("Close");
HBox hbox = new HBox(5, okBtn, closeBtn);
root.getChildren().addAll(hbox);
AnchorPane.setRightAnchor(hbox, 10d);
 AnchorPane.setBottomAnchor(hbox, 10d);
Scene scene = new Scene(root, 300, 200);
stage.setTitle("AnchorPane Ex");
stage.setScene(scene);
 stage.show();
```

Gestionarea pozitionarii componentelor UI▪ AnchorPane
StudentView1.java Example

```
private Node initTop() {
AnchorPane anchorPane=new AnchorPane();
Label l=new Label("Student management System");
l.setFont(new Font(20));
AnchorPane.setTopAnchor(l,20d);
AnchorPane.setRightAnchor(l,100d);
anchorPane.getChildren().add(l);
Image img = new Image("logo.gif");
ImageView imgView = new ImageView(img);
imgView.setFitHeight(100);
imgView.setFitWidth(100);
imgView.setPreserveRatio(true);
AnchorPane.setLeftAnchor(imgView,20d);
AnchorPane.setRightAnchor(imgView,10d);
anchorPane.getChildren().add(imgView);
return anchorPane;
}
```

Gestionarea pozitionarii componentelor UI ▪ GridPane

```
GridPane gr=new GridPane();
gr.setPadding(new Insets(20));
gr.setAlignment(Pos.CENTER);
gr.add(createLabel("Username:"),0,0);
gr.add(createLabel("Password:"),0,1);
gr.add(new TextField(),1,0);
gr.add(new PasswordField(),1,1);
Scene scene = new Scene(gr, 300, 200);
stage.setTitle("Welcome to JavaFX!!");
stage.setScene(scene); stage.show();
```

Lucrare de laborator nr. 6

1. Tema lucrării:

Crearea interfețelor grafice în baza tehnologiei JavaFX

2. Scopul lucrării:

Însuşirea modalităţilor de creare şi realizare a inerfețelor în Java utilizînd tehnologia JavaFX;

3. Etapele de realizare:

1) Crearea interfeţii programului;
2) Crearea obiectelor grafice;
3) Prezentarea lucrării.

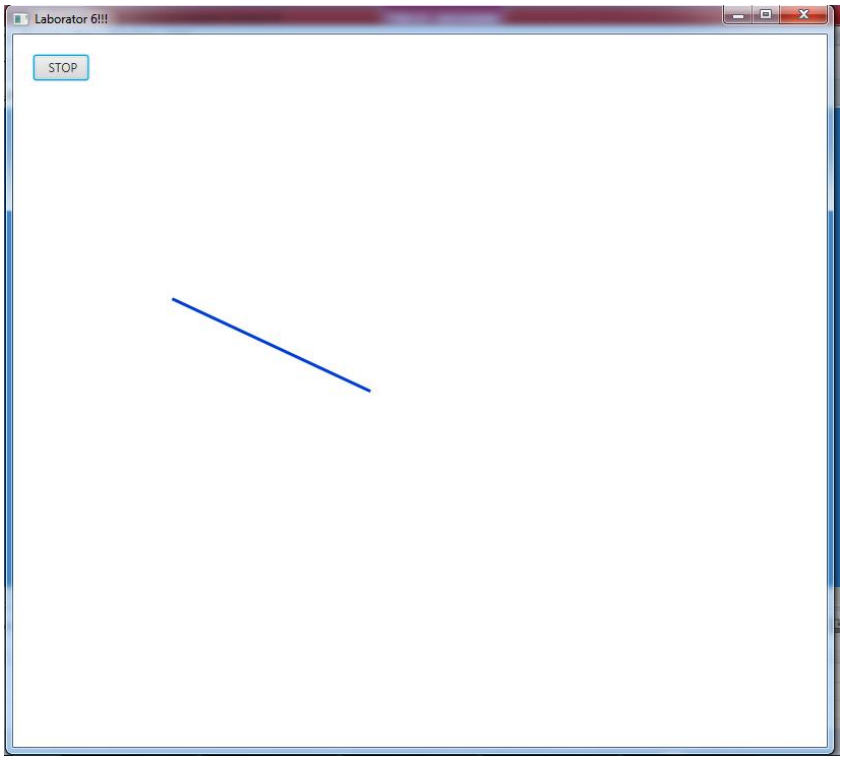
4. Exemplu de realizare:

```
import javafx.application.Application;
```

```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.control.Button;
import javafx.animation.Timeline;
import javafx.animation.KeyFrame;
import javafx.util.Duration;
import javafx.scene.shape.Line;
import javafx.scene.transform.Rotate;

public class Main extends Application {

	int unghiul = 0;
	int red;
	int green;
	int blue;
	static int col = 0;
	double x = 500, y = 500, xf = 350, yf = 350;

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

	@Override
	public void start(Stage primaryStage) {
		primaryStage.setTitle("Laborator 6!!!");

		Line line = new Line(xf, yf, 200, 200);
		line.setStroke(Color.RED);
		line.setStrokeWidth(3);

		Button button = new Button("START");
		button.setTranslateX(20);
		button.setTranslateY(20);

		Group root = new Group();
		Canvas canvas = new Canvas(800, 700);

		Timeline timeline = new Timeline();
		timeline.setCycleCount(Timeline.INDEFINITE);
		timeline.setAutoReverse(true);
		timeline.getKeyFrames().addAll(new KeyFrame(Duration.millis(100), ae ->
drawShapes(line)));

		button.setOnAction(event -> {
			if (col == 0) {
				timeline.play();
				button.setText("  STOP ");
				col = 1;
			} else {
				timeline.stop();
				col = 0;
				button.setText(" START ");
```

```
                }

            });
            root.getChildren().add(line);
            root.getChildren().add(canvas);
            root.getChildren().add(button);
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        }

        void drawShapes(Line line) {

            int red;
            int green;
            int blue;

            red = (int) (Math.random() * 256);
            green = (int) (Math.random() * 256);
            blue = (int) (Math.random() * 256);
            line.setStroke(Color.rgb(red, green, blue));
            line.setStrokeWidth(3);

            line.getTransforms().add(new Rotate(20, xf, yf));

        }
```

Rezultatul realizării:



5. Probleme propuse:

Pentru toate variantele de criat o interfață grafică cu minim 3 componente standarte și minim 2 componente grafice proprii care realizează sarcina indicată.

1. Sa se creeze mişcare pe ecran a şirurilor de caractere (unul după altul) dintr-un tablou de şiruri. Direcţia de mişcare şi consecutivitatea fiecărui rând este selectat aleatoriu.
2. De creat 3 primitive grafice ecran, care apar din diferite parti ale ferestrei, şi sa se alinieze unul după altul în centru. Procesul trebuie să fie repetat ciclic.
3. Sa se creeze o mişcare a cercului pe ecran, astfel încât, atunci când atingeţi limitele ecranului, cercul este reflect, cu efect de comprimare elastică. Mișcarea este continuă pînă cînd nu va atinge linia din fereastră care nu se mișcă. Locația ei este aliatoare.
4. De prezentat într-o fereastră apropierea mingei şi deplasarea ei. Mingea trebuie să se mişte, cu o viteza constantă. La cerere începe mișcarea din nou.
5. Sa se reprezinte în fereastră un segment, ce se roteste în planul ecranului în jurul unuia din punctele finale ale lui. Segmentul trebui să-şi schimbe culoarea de la un pas la altul.
6. Sa se reprezinte în fereastră un segment, ce se roteste în planul ecranului în jurul punctului, care se deplasează de-a lungul segmentului. Să poată fi startar și restartat.
7. Să se reprezinte un dreptunghi, care se roteste în planul ecranului în jurul centrului său de greutate. Să poată fi startar și restartat.
8. Să se reprezinte un dreptunghi, ce se roteste în planul ecramului în jurul unuia dintre virfuri. Să poată fi startar și restartat.
9. Reprezintați o elipsă, ce se roteste în plan ecranului în jurul centrului său de greutate. . Să poată fi startar și restartat.
10. Sa se creeze mişcare pe ecran a imaginilor dintr-un directoriu. Direcţia de mişcare pe ecran şi consecutivitatea fiecării imagini este selectată aleatoriu.
11. Sa se reprezinte o imagine cu un text informațional. Textul poate fi modificat.
12. Sa se reprezinte o fereastră cu calendarul lunii curente.