CAPITOLUL 5

3.1 Utilizarea tread-urilor în grafică

## Un program care conține obiecte grafice folosește concurența pentru creare ainterfeței utilizator ce va raspunde integrat cu utilizatorul, indiferent de procesele, pe care le execută. Pentru aceasta trebuie de știut cum tratează framework-ul grafic, firele de execuție. În graphică apar urmatoarele tipuri de fire de execuție:

## – Fire de execuție inițiale, ce rulează codul inițial al aplicației.

## – Firul de execuție rezervat expedierii evenimentelor. Majoritatea codului de interacțiune cu framework-ul grafic trebuie executat pe acest fir.

## –Fire de execuție background, sunt acele fire ce execută în afara aplicației operațiile ce necesită mult timp de procesare.

##### Lucrarea de laborator nr. 5

1. Tema lucrării:

Animaţia scenelor cu ajutorul thread-urilor

2. Scopul lucrării:

* Însuşirea modalităţilor de creare a tread-urilor în scena grafică în Java;
* Însuşirea metodelor de utilizare a thread-urilo rîn scena pentru animții.

3. Etapele de realizare:

1. Utilizarea clasei Thread pentru crearea unei clase noi;
2. Utilizarea interfeţei Runnable pentru crearea şi lansarea de thread-uri;
3. Utilizare a firelor de execuție pentru animaţia scenei grafice.

**4. Exemplu de realizare:**

importjava.awt.BorderLayout;

importjava.awt.Color;

importjava.awt.Dimension;

importjava.awt.Graphics;

import java.awt.Graphics2D;

importjava.awt.Image;

importjava.awt.Point;

importjava.awt.RenderingHints;

importjava.util.Calendar;

importjavax.swing.JFrame;

importjavax.swing.JPanel;

importjava.awt.event.MouseAdapter;

importjava.awt.event.MouseEvent;

importjava.awt.image.BufferedImage;

importjava.io.File;

importjava.io.IOException;

importjava.awt.FlowLayout;

importjavax.imageio.ImageIO;

importjavax.swing.JButton;

public class Clock extends JFrame {

private static final long serialVersionUID = 1L;

private static final Color BACKGROUND\_COLOR = new Color(24, 116, 205);

public Clock() {

 getContentPane().setBackground(Color.DARK\_GRAY);

ClockPanel container = new ClockPanel();

container.addMouseListener(new MouseAdapter() {

 @Override

 public synchronized void mouseClicked(MouseEvent arg0){

 System.out.println("Click! " + container.b);

 if(container.b == true) container.b = false;

 else if(container.b == false) container.b = true;

 notifyAll();

 }

 });

getContentPane().setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

container.setBackground(Color.DARK\_GRAY);

getContentPane().add(container);

setBackground(Color.DARK\_GRAY); // setUndecorated(true);

pack();

setVisible(true);

 }

public static void main(String[] args) {

new Clock();

 }

}

classClockPanel extends JPanel implements Runnable {

private static final long serialVersionUID = 1L;

public Thread t = new Thread(this);

public static boolean b = true;

 /\*\* The coordinates used to paint the clock hands. \*/

intxHandSec, yHandSec, xHandMin, yHandMin, xHandHour, yHandHour;

 /\*\* The size of the clock. \*/

private final int HORIZONTAL\_SIZE = 500;

private final int VERTICAL\_SIZE = 500;

 /\*\* The length of the clock hands relative to the clock size. \*/

private final intsecondHandLength = HORIZONTAL\_SIZE / 2 - 50;

private final intminuteHandLength = HORIZONTAL\_SIZE / 2 - 70;

private final inthourHandLength = HORIZONTAL\_SIZE / 2 - 100;

 /\*\* The distance of the dots from the origin (center of the clock). \*/

private final int DISTANCE\_DOT\_FROM\_ORIGIN = HORIZONTAL\_SIZE / 2 - 40;

private final int DIAMETER\_BIG\_DOT = 8;

private final int DIAMETER\_SMALL\_DOT = 4;

private final static Color GREY\_COLOR = new Color(160,160,160);

privateBufferedImagemyImage;

publicClockPanel() {

setMinimumSize(new Dimension(HORIZONTAL\_SIZE, VERTICAL\_SIZE));

setMaximumSize(new Dimension(HORIZONTAL\_SIZE, VERTICAL\_SIZE));

setPreferredSize(new Dimension(HORIZONTAL\_SIZE, VERTICAL\_SIZE));

setLayout(null);

try {

 myImage = ImageIO.read(new File("src/psa-logo.png"));

 } catch (IOException e) {

 // TODO Auto-generated catch block

 e.printStackTrace();

 }

t.start();

 }

 /\*\*

\* At each iteration we recalculate the coordinates of the clock hands,

 \* and repaint everything.

 \*/

public void run(){

while(true){

try{

intcurrentSecond = Calendar.getInstance().get(Calendar.SECOND);

intcurrentMinute = Calendar.getInstance().get(Calendar.MINUTE);

intcurrentHour = Calendar.getInstance().get(Calendar.HOUR);

xHandSec = minToLocation(currentSecond, secondHandLength).x;

yHandSec = minToLocation(currentSecond, secondHandLength).y;

xHandMin = minToLocation(currentMinute, minuteHandLength).x;

yHandMin = minToLocation(currentMinute, minuteHandLength).y;

xHandHour = minToLocation(currentHour \* 5 + getRelativeHour(currentMinute), hourHandLength).x;

yHandHour = minToLocation(currentHour \* 5 + getRelativeHour(currentMinute), hourHandLength).y;

repaint();

Thread.sleep(500);

 } catch(InterruptedExceptionie){

ie.printStackTrace();

 }

 }

 }

privateintgetRelativeHour(int min) {

return min / 12;

 }

protected void paintComponent(Graphics g){

 Graphics2D g2 = (Graphics2D)g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.clearRect(0, 0, getWidth(), getHeight());

g2.drawImage(this.myImage, 0, 0, this);

 // Draw the dots

if(this.b == true){

g2.setColor(GREY\_COLOR);

for (inti = 0; i< 60; i++) {

 Point dotCoordinates = minToLocation(i, DISTANCE\_DOT\_FROM\_ORIGIN);

g2.setColor((i<= Calendar.getInstance().get(Calendar.SECOND)) ? Color.black : GREY\_COLOR);

if (i % 5 == 0) {

 // big dot

g2.fillOval(dotCoordinates.x - (DIAMETER\_BIG\_DOT / 2),

dotCoordinates.y - (DIAMETER\_BIG\_DOT / 2),

 DIAMETER\_BIG\_DOT,

 DIAMETER\_BIG\_DOT);

 } else {

 // small dot

g2.fillOval(dotCoordinates.x - (DIAMETER\_SMALL\_DOT / 2),

dotCoordinates.y - (DIAMETER\_SMALL\_DOT / 2),

 DIAMETER\_SMALL\_DOT,

 DIAMETER\_SMALL\_DOT);

 }

 }

 // Draw the clock hands

g2.setColor(Color.black);

g2.drawLine(HORIZONTAL\_SIZE / 2, VERTICAL\_SIZE / 2, xHandSec, yHandSec);

g2.drawLine(HORIZONTAL\_SIZE / 2, VERTICAL\_SIZE / 2, xHandMin, yHandMin);

g2.drawLine(HORIZONTAL\_SIZE / 2, VERTICAL\_SIZE / 2, xHandHour, yHandHour);

 }

}

private Point minToLocation(inttimeStep, int radius) {

double t = 2 \* Math.PI \* (timeStep-15) / 60;

int x = (int)(HORIZONTAL\_SIZE / 2 + radius \* Math.cos(t));

int y = (int)(VERTICAL\_SIZE / 2 + radius \* Math.sin(t));

return new Point(x, y);

 }

}

Rezultatul realizării programului:



**5. Probleme propuse spre realizare:**

De scrisun program pentrurealizareaunei scene cevautiliza minim treifirele de execuţie pentru realizare:

* 1. Ceas, cepoate fi setat şi restartat prin click ( prefaţa unei emisiuni TV);
	2. Cronometru, cu butoanele start/stop şi reset;
	3. Generare de figuri geometrice ce se mişcă, ce pot fi create sau distruse apăsând butonul ce corespunde figurii;
	4. De desenat un derptunghi ce se deplasează pe ecran și schimbă culoarea tastînd un buton oarecare sau cu mausul;
	5. Componenta text ce îşi schimbă umbrele în dependenţă de mişcarea sursei de lumină;
	6. Image loader ce afişează un şir de imagini, în timp ce se poate de scris text în TextArea;
	7. Spot publicitar, ce poate fi întrerupt şi repornit, ce caracterizează FCIM;
	8. Spot publicitar ce poate fi întrerupt şi repornit, ce caracterizează Departamentul de Informatică și Ingineria Sistemelor;
	9. Spot publicitar ce poate fi întrerupt şi repornit, ce caracterizează Cursul Programarea concutentă;
	10. Componentă, ce poate fi întreruptă şi repornită, ce reprezintă mişcarea cu viteze diferite a unui grup de cercumferințe.

**6. Întrebări de verificare:**

1. **Avantajele utilizăriifirelor de execuție în aplicațiile grafice Java?**
2. **Definiți firul de execuție inițial?**
3. **Definiți firul de execuție rezervat?**
4. **Definiți firul de execuție background?**